EVENT BUILDER & LEVEL 3

MANUAL

Expert's Guide

Nuno Leonardo

Massachusetts Institute of Technology

for the EVB+LEVEL3+ONLINE teams

October 2003


HEAT EXCHANGER
HEAT EXCH
COF FAN PACK

CDF/DOC/ONLINE/PUBLIC/6138
Original version: August 1, 2002
Update version: November 2, 2003

Event Builder and Level 3 Manual

K. Anikeev, G. Bauer, A. Belloni, A. Bolshov, I. Furić,
G. Gómez-Ceballos B. Iyutin, T.H. Kim, B. Knuteson, A. Korn,
I. Kravchenko, N. Leonardo, J. Miles, M. Mulhearn, M. Neubauer,
Ch. Paus, A. Rakitin, S. Tether, J. Tseng, F. Würthwein
Massachusetts Institute of Technology

Abstract

The set of topics presented roughly defines the scope of knowledge that is necessary for EVB/L3 pager duties. The items listed under "advanced" category can be skipped on the first stages of getting familiar with the system but will likely be needed if a serious problem occurs.

Foreword

The Event Builder (EVB) and Level3 (L3) systems constitute a crucial component of the data acquisition structure of CDF. They are responsible for collecting and assembling together the event fragments from the various Front End crates, organizing the disordered event data pieces fully reconstructing the event, making a final trigger decision, and transferring the accepted events to CSL towards offline storage.

The purpose of this document is to provide a *working knowledge* of the EVB and L3 systems of CDF at Run II. It constitutes a *functional*, in addition to *operational*, description of the two systems, and of how they are integrated together and in the full DAQ system of the experiment.

It should give a detailed enough overview of the systems so as to provide in general a comprehensive source of information for EVB and L3 training – namely, for experts on call. It is also structured in a convenient way – each main topic is presented as a set of various sub-topics, which themselves are decomposed into several paragraphs. This way it should as well constitute a easy-to-look reference.

Nuno[1] and the EVB/L3 Team

[1]mailto: leonardo@fnal.gov

Contents

List of Figures

Thematic Index

Hardware and Operating System

HARDWARE
&
OPERATING SYSTEM

1 Hardware and Operating System

1.1 Necessary Linux knowledge

Linux-i **How to check Linux flavors on L3 farm and why they are not uniform**

Currently we are using two versions of Linux kernel (*flavors*) [2] on the farm. This can be checked using the Linux command **uname** for printing system information. For example, as of the writing, the following is displayed for the two Gateways:

```
b0l3gate1> uname -rs
b0l3gate1> Linux 2.0.38

b0l3gate2> uname -rs
b0l3gate2> Linux 2.2.14-1.3.0f2smp
```

The reason for this non uniformity is that the driver for the ATM switch was written for an original flavor and has not been updated. For this reason, the Converters are using this older flavor, while the Processor and Output nodes use an updated one. Note that this requires that code compilation be performed in both Gateways.

Linux-ii **How to use man**

The Linux manual pages for a command are displayed using **man** *name-of-command*.

The flag **k** e.g. prints one line descriptions for all man pages related to a specified *keyword* (same as **apropos**)

Linux-iii **How to check for running processes**

Process status is reported using the **ps** command. By default the **ps** program only shows processes that maintain a connection with a terminal; processes that run without communicating with a user on a terminal can be see with the **x** flag.

In particular

ps -auxwww

shows processes of all users (**a**) identifying them (**u**) without controlling terminal (**x**), and each flag **w** specified will add another possible line to the output.

ps -l displays also a NI (nice) column with the *priority* values of the active processes. [3]

[2] In general, **flavor**s are used to indicate the operating system, or OS version, dependency of a product.
[3] The priority of a process can be set with **nice** *command* and adjusted with **renice**.

Linux-iv **How to check memory and CPU usage**

Top CPU processes are displayed with **top** command.

Linux-v **How to grep through log files**

grep (general regular expression program) is a very powerful *search* tool. It prints lines matching a specified pattern; for example

grep *string* *file*

returns all the lines that contain a string matching the expression *string* in log file *file*. The flag **i** ignores capitalization; **v** prints all lines not matching *string*.

Additionally, instead of having it searching directly a file, *grep* can accept data through STDIN, by redirecting the output of another command using the pipe [4] (|) operator, as in

cat *file* | **grep** *string*

Linux-vi **Useful commands**

Some few commands may reveal particularly useful.

In order to search for a file it can be used commands like

- **whereis** locate the binary, source, and manual page files for a command.
- **locate** list files in databases that match pattern.
- **find** search for files in a directory hierarchy.

There are however differences. **whereis** will search only particular paths to find binaries and or manpages (manpages tell where it looks). **locate** uses a database created by an updatedb to efficiently locate files. It works great, assuming database is updated often enough, but need to be careful otherwise. One other possibility is to use **find** , which will search each and every path recursively from it's start point. It will in principle be slower than the previous two, but is most powerful. E.g., the following

```
find /home -name core -exec rm{} \;
```

would find all core files in home directories and remove them.

- **top** give continual reports about the state of the system, including a list of the top CPU using processes

The following are file pagers, programs that display text files.

[4] *Pipes* connect processes together, allowing the output of a program be used as the input of another one.

- **tail** delivers the last part of the file; the flag **-f** *follow*, attempts to read and copy further records from the input-file
- **less** displays the contents of files

The most frequently used **less** commands are *Space*, *Return*, *b*, /, ?, and *q*, for scrolling and searching, forwards and backwards.

Both **tail -f** and **less +F** may be used to monitor the growth of a file that is being written by some other process.

Linux-vii **What is nfs-mounted disk, how to check which disks are seen on a node**

The Network File System (NFS) was developed to allow machines to mount a disk partition on a remote machine as if it were on a local hard drive. This allows for fast, seamless sharing of files across a network.

The file **/etc/fstab** contains static information about the filesystems. Type **man fstab** for information on each field in the file.

In particular it describes what devices are usually mounted (type **man mount** to see how), including during booting.

Linux-viii **How to check available disk space, used disk space**

df displays the number of total bytes on a partition, the number used, and the percentage of space used. The flag **h** displays output is a more human friendly form (appends M$\sim$ Meg, G$\sim$Gig).

du displays the space used by each directory in the given path. Flag **a** summarize disk usage of each file, recursively for directories, and **s** provides a grand total for the directory.

mount displays all mounted devices, their mount-point, filesystem, and access (with command line arguments is used to mount file system)

Linux-ix **What is CVS, how to checkout, commit, find diff, update or check the status of a package**

CVS (*Concurrent Versions System*) is a version control system.

It stores all the versions of a file in a single file in a clever way that only stores the differences between versions. All files are stored in a centralized *repository*, which – in the case of the cdf online code – is located in **b0dau30**

```
b0l3gate1> echo $CVSROOT
b0l3gate1> cvs@b0dau30.fnal.gov:/cdf/code/cvs
```

A typical work-session includes the following steps:

- Get source

  cvs checkout *module*

  This creates a private copy of the source for *module* in a directory with that name. The files in the directory **./***module*/**CVS/** are used internally by **CVS** and should not in principle be modified.

- Commit changes

  cvs commit *file*

  This will store the modified *file* in the repository.

- Clean up

  In the end, the working copy can be erased simply by removing the directory created originally by **cvs checkout.** More convenient however, is to use the **cvs release** command, that checks that no uncommitted changes are present, and in that case the **d** flag removes the working copies.

- View differences

  cvs diff [**-r** *rev1*] [**-r** *rev2*] *file*

  cvs diff shows differences between files in working directory and source repository, or between two revisions in source repository.

  If a particular revision is not specified, the files are compared with the revision they were based on.

  If no files are specified, it will display differences for all those files recursively in the current directory that differ from the specified or corresponding revision in the source repository.

cvs update updates copies of source files from changes that other developers have made to the source in the repository.

cvs status shows current status of files – latest version, version in working directory, whether working version has been edited.

Linux-x **What is ups and what is setup command for**

UPS (*Unix Product Support*) [5] is a toolkit developed at Fermilab designed for the management of software products on local systems, and to facilitate the product distribution and configuration management tasks of the product providers.

Once one has an UPS product, all the settings necessary for it to work properly are done automatically via the command

setup *[product-name]*

[5] Detailed information at http://www.fnal.gov/docs/products/ups/.

What this does is to execute the script *setups.csh*, which in particular sets/modifies environment variables: PRODUCTS (points to database), UPS_DIR (points to product root directory), PATH (modified to include $UPS_DIR/bin), SETUP_UPS (contains information for *unsetup* command).

All the other ups commands (other than *un*setup) have the standard syntax

ups *command [options] product-name [version]*

and include the following.

ups flavor returns flavor information.

ups list returns information about the declared product instances in a UPS database. It displays what products are in the database, what the current version of a product is for the machine's flavor and what other versions might be available.

ups modify allows you to manually edit any of the database product files. It performs syntax and content validation before and after the editing session. **ups verify** checks the integrity of the database files for the specified product(s), and lists any errors and inconsistencies that it finds.

A list of all the UPS commands with brief definitions is displayed with the **ups help** command.

In the Gateways, the location of a given ups product has the form

/usr/products/*product-name*/*flavor*/*product-version*/

The ups table for each product, used e.g. by the **setup** command, is located at

/usr/products/*product-name*/*flavor*/*product-version*/ups/*product-name*.table

How to transform a product into an ups product?

Product declaration is done with the **ups declare** command.

If the product was installed with **upd install** [6] , this automatically makes the initial declaration of the product to the local UPS database. Otherwise the product need to declare it manually, which is done with the **ups declare** command. The following is a common syntax

```
ups declare product-name version -r /path/to/prod/root/dir/
   -f flavor [-z /path/to/database] [-U /path/to/ups/dir]
   [-m table_name.table] [-M /path/to/table/file/dir] [chainFlag]
```

The steps in producing an ups product could involve:

[6] See indicated ups reference above.

1. *get and compile the source* - upd install; cvs checkout, make
2. move to proper place in /usr/products/... directory
3. create/edit ups table
4. declare product as ups as in the example that follows

```
setenv ace /usr/products/cdfevb/ace/v2_14/
ups declare  -z /usr/products/upsdb  -r $ace  -m cdfevb_ace.table
 -s NULL  -U $UPS_DIR  -M $ace/ups  cdfevb_ace  v2_14
setup cdfevb_ace [v2_14]
```

In order to declare a specific version of a product current, still taking the previous example, type

```
ups declare -g current cdfevb_ace v2_14 -z /usr/products/upsdb
```

Linux-xi **What are semaphores, message queues and shared memory (on conceptual level)** [7]

These are three interprocess communication (**IPC**) facilities.

Semaphores are data structures that are used for synchronization between two or more processes. Basically they can be viewed as a single integer that represents the amount of resources available. When a process wants a resource, it checks the value of the semaphore, and if it is nonzero, it decrements the appropriate number from the semaphore in accordance to the amount of resources it wishes to use. The kernel will block the process if the semaphore is zero or doesn't have a value high enough for the decrement.

Message queues provide a memory based FIFO between two processes. The primary difference between a message queue and a socket or named pipe is that message queues may have multiple processes reading and writing from and to them, or no readers at all.

Shared memory (SHM) is a method of IPC whereby two or more unrelated processes can access the same logical memory. It provides a very efficient way of sharing and passing data between multiple processes.

Linux-xii **tcsh and csh scripts, how to write a simple script, where is documentation on scripting**

Shell scripting is central in Linux, and essential for simple system administration tasks. It is very well documented on the web. [8]

The simplest possible script is

[7] See e.g. the book [[5]] *Beginning Linux programming*, chapter 12, on the trailers.
[8] See e.g. http://www.linuxdoc.org/LDP/abs/ for bash scripting.

```
#!/bin/bash
echo "hello world\n"
exit 0
```

Basic knowledge of other scripting languages (e.g., sed, awk, perl, ...) may also reveal extremely useful for specific tasks.

Advanced ... but not too much

Linux-xiii **How to check system resources with ipcs and to clear them with ipcrm**

ipcs Report interprocess communication facility status for queues, shared memory, semaphores:

ipcs -s list Semaphores

ipcs -q list share queues

ipcs -m shared memory

ipcrm Remove queues, shared memory, or semaphores:

ipcrm -sem *semid* list Semaphores

ipcrm -msg *msqid* list share queues

ipcrm -shm *shmid* shared memory

Linux-xiv **How to check connections between programs and computers with netstat**

netstat displays information of the Linux networking subsystem – network connections, routing tables, interface statistics, masquerade connections, netlink messages, and multi- cast memberships.

Linux-xv **Expect**

Expect is a Unix tool for automating interactive applications.

In a way, Expect automates a software dialogue – it reads a script that resembles the dialog itself, and by following the script it knows what can be expected from a program and what the correct responses should be. The script can specify responses by patterns, and can take different actions on different patterns.

The three basic commands are *send* (sends strings to a process), *expect* (waits for strings from a process), and *spawn* (starts a process).

The simplest Expect script is

```
#!/usr/bin/expect -f
send "hello world\n"
exit
```

Expect is a tool also employed on Level3, e.g. allowing to execute commands over specified sets of nodes. [9] Tutorials are also available on the web. [10]

Linux-xvi **Root password, how to shut down and reboot Linux node properly**

/sbin/shutdown -h Halt after shutdown

/sbin/shutdown -r Reboot after shutdown (or **reboot**)

Linux-xvii **What can be found from: /proc, /etc/hosts, and /var/log/messages**

/proc is a pseudo-filesystem which is used as an interface to kernel data structures. It shows the running processes. Commands like e.g. **ps** and **top** use information stored in these files. Follow examples of files in */proc*.

/proc/stat contains status information about the process;

/proc/cmdline holds the complete command line for the process;

/proc/meminfo reports the amount of free and used memory;

/proc/cpuinfo contains information of (each) CPU;

/proc/modules lists of the modules that have been loaded by the system, e.g. in a Converter

```
b0l3c02> more /proc/modules
scramnet          1          3
nicstar1_5        4          0
tulip             7          2
```

where *scramnet, nicstar, tulip* are the modules correspondent to SCRAMNet, ATM, Ethernet networks, resp.;

/proc/atm/devices is another relevant file for detecting problems related to ATM connections. [11]

/etc directory contains critical startup and configuration files. The */etc/hosts* file (locally) maps names to IP addresses. Each line starts with an IP address and continues with the various symbolic names by which that address is known. The first line in particular should be

```
127.0.0.1               localhost localhost.localdomain
```

[9] As a single example, the Expect script $\$L3_{CSHDIR}/l3\_all\_nodes\_exe$ can be used (from Gateway2 as user l3proxy) to execute a specified shell command on all nodes of the farm.

[10] E.g. http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/expect/.

[11] See later section.

/var/log directory contains various system log files. */var/log/messages* holds messages printed to console ... (e.g., last time of reboot, ...)

Linux-xviii **Cron scripts that keep disk space usage low (Gateways, Converters, other nodes)**

Periodic execution under Linux is handled by the **cron** deamon. cron starts when the system boots and remains running as long as the system is up.

Scripts located in directories **/etc/cron.daily, cron.hourly/, cron.monthly/, cron.weekly/** are executed at correspondent times.

In particular, maintenance of log files (and eventually executable versions) are automated with cron.

As an example, */etc/cron.daily/tmpwatch* removes files which haven't been accessed for a period of time.

1.2 VME standard

VME-i **What is VME**

VME, which stands for *Versa Module Eurocard*, is a bus system. In general a bus is a common channel between multiple devices, that allows different pieces of hardware to communicate with each other.

VME uses a master-slave architecture – functional modules called masters transfer data to and from functional modules called slaves. In general, a master is able to initiate data transfer cycles, whereas a slave detects bus cycles generated by masters, and participates in the cycles if they are selected. VMEbus allows in general multiple masters to share the data transfer bus, thereby creating a multiprocessor system.

Data transfer is *asynchronous*.[12]

VME-ii **How does VME crate looks like**

VME crates are located on the first floor of B0 building (and another in the 3rd). There one finds non-silicon (CPUs denoted b0eb11 to b0eb16) and silicon (CPUs denoted b0eb17 to b0eb25; b0eb19 being the SRC board, responsible for silicon trigger) and, on the third floor, the Scanner Manager (CPU denoted b0eb10) crates.

VME card cages contain 21 slots. They contain several boards — the bus master, SCRAMNet adapter, and various *VME Readout Board*s (VRBs).

VME-iii **What is backplane, what is it for**

VME backplane is an interior part of the VME crate where the various bus cards are connected to. It is really the implementation of the bus itself.

Mildly advanced

VME-iv **Power supplies, what voltages are needed**

The power suppliers are located on the bottom of the rack, below the VRB crates. The lower(top) one feeds the lower(top) crate.

Non-silicon and silicon crates use different voltages on the suppliers – indicated by the supplier green led; needed voltages are $-5.2V$, $12V$, $5V$ and $\pm 12V$, $5V$, $3.3V$, resp.

The SCPU boards for non-silicon crates have a transformer incorporated. [13]

[12] PCI on the other hand is an example of a *synchronous* bus, as data transfer is coordinated by a clock.
[13] located on the SCPU adapter, is used for supplying transition module 712 with $-12V$.

VME-v **What is 6U or 9U, what is an adapter, where are the fuses on adapters and how to check and replace them**

There are different size VME crates: 6U (smaller) and 9U (bigger) (6 and 9 standard units, resp.). E.g. the Scanner Manager (b0eb10, in 3rd floor) connects directly to a 6U crate, but the SCPUs belong to 9U crates, and so actually they are connected to an adapter, which itself connects to the cage. I.e., an adapter allows for incorporating smaller 6U type boards into bigger 9U type crates.

The fuses are located on the adapters. There are four of them (can be recognized as resistor-size components, of green color).

There are three red leds associated with the fuses, which one can try to confirm are on even with the adapter inserted in the cage (provided the slot on its rhs is empty).

The way to check them involves [14] turning off the correspondent power supply, removing the adapter from the cage, localizing the components themselves and check them individually with an amperimeter.

In case they are burnt, it's necessary to remove (pull out) and replace them.

VME-vi **What is a bus master, how to make a CPU to be one**

Bus master is the crate manager, a local VME-based processor.

Each SCPU crate possesses two CPU units, the bus master and an SCPU. (Additionally, each VRB board possesses a lower level processing unit.)

If needed, an SCPU can be made the master. That involves changing the position of a *jumper* (red color). Jumper sites are located near the edge of the SCPU board (not adapter), and are labeled as *J#*. For the present purpose the relevant one is *J22*.

There are three possible positions: no jumper (default; board is not master) and with jumper positioned in each of the two end pairs (out of the three) of pins (board is bus master, permanently or automatically[15]).

[14] After notifying the Control Room.
[15] I.e., when it detects it is necessary.

1.3 VxWorks

VxWorks-i **What is VxWorks**

VxWorks is a Unix-like operating system for embedded real-time applications. It is 'Unix-friendly' — it inter-operates nicely with the Unix environment and provides for many of the Unix C-library routines, like file I/O, sockets, RCP, etc.

It is the standard real-time operating system in the embedded VME processors at CDF (?). Of particular interest is that the SCPUs, and the Scanner Manager, run under VxWorks.

VxWorks is a *real-time* OS – the programmer can have control on the way the tasks are executed (scheduling). It is a 'preemptive' *multi-task* OS. Each task has a priority level. Supposing that the CPU is running a task t1 of priority p1, and a task t2 of priority p2>p1 is called, then t1 is suspended right away and t2 is executed. When a resource needed for t2 is missing, t2 is turned in "pending" state. If the execution of t2 has been completed, t2 is discarded from the schedule. In either case the CPU will resume the execution of t1, unless a task t3 of priority p3 such that p3<p2 and p3>p1 has been spawned by p2.

When two tasks with the same priority level must be managed, the CPU will spend alternatively a certain amount of time ("time slice") on each task until they are completed. The time slice length can be defined by the programmer.

VxWorks supports concurrent tasking–workstations communicating with each other by exchanging messages. VxWorks is multi-threaded and supplies all intertask messaging functions, including semaphores, pipes, sockets, and TCP/IP interprocessor communications.

VxWorks supports disk- and network-based filesystems.

VxWorks supports remote debugging from workstations.

VxWorks-ii **How to log in to VxWorks** [16]

To log in a VxWorks board, one can use one of the commands **vxlogin**, **rlogin**, **telnet**. Exiting is with **logout** command.

E.g., logging from Gateway1 into one of the SCPUs

```
b0l3gate1> rlogin b0eb11
-->
   ...
--> logout
b0l3gate1>
```

If it is not possible to rlogin, one should try to connect via *minicom*. [17]

[16] See http://www-b0.fnal.gov:8000/vxworks/vxworks.html

[17] If that still doesn't work it may be necessary to check the crate, and eventually reboot it.

```
b0dau30~ > ssh b0dap10
<b0dap10.fnal.gov> minicom b0eb11
  ...
ctrl-A Q Enter
```

VxWorks-iii **How to reboot vxworks, when does interrupted boot prompt appears**

Rebooting is performed with the **reboot** command.

There should be no SUSPENDed tasks. If that is however the case, reboot may be necessary.

It may happen that the usual (− >) VxWorks prompt does not show up after trying to log in, and instead appears an *interrupted prompt* like

```
[VxWorks]:
```

In this case, the command for rebooting is still *ctrl-X* and may be particularly useful. It is also still possible to change configuration variables on this prompt.

VxWorks-iv **How to check and change vxworks boot parameters, what are boot/startup scripts and where they are found**

VxWorks uses some variables for boot configuration from vxboot [18] node. Those variables can be viewed and modified with the command **bootChange.** For example,

```
-> bootChange

'.' = clear field;  '-' = go to previous field;  ^D = quit

boot device          : dc
processor number     : 2
host name            : b0l3boot-vx
file name            : $VXWORKS_MV2604
inet on ethernet (e) : 192.168.20.11:ffffff00
inet on backplane (b):
host inet (h)        : 192.168.20.130
gateway inet (g)     : 192.168.20.11
user (u)             : vxboot
ftp password (pw) (blank = use rsh):
flags (f)            : 0x0
```

[18] This is a Linux machine, used by SCPUs and SM to boot from.

```
target name (tn)      : b0eb11
startup script (s)    : $SCPU_ALL_SCRIPT
other (o)             : mv2604

value = 0 = 0x0
->
```

The are *startup scripts* that are executed on booting, and that are defined by *$SCPU_ALL_SCRIPT*. They are located on vxboot node; see its *.chsrc* file.

VxWorks-v **What are most useful commands on VxWorks**

To learn about the different commands, type **help** on VxWorks prompt.

```
--> help
  ...
help                            Print this list
netHelp                         Print network help info
h          [n]                  Print (or set) shell history
i          [task]               Summary of tasks' TCBs
ti         task                 Complete info on TCB for task
td         task                 Delete a task
ts         task                 Suspend a task
tr         task                 Resume a task
  ...
```

Important commands include the following.

- **i** *[task]* displays tasks currently managed by the VxWorks system; among the displayed parameters are the name, the priority level (0 for the highest priority, 255 for the lowest), the task ID (in hexadecimal), and the status of the task (pending, ready (=running), suspended).
- **memShow**
- **ti** *task* complete info for task
- **tt** *task* prints the stack trace of a task.
- **moduleShow**
- **hostShow**
- **taskDelay** *task*
- **td** *task* aborts a task
- **ts** *task* suspends a task; unlike td, the task will still be displayed in the list by **i**
- **tr** *task* resumes a suspended task; the system will resume the execution of the task at the point it was suspended

- **reboot** the same as *ctr-X*
- **logout**

task is either the name or the task ID (with the 0x prefix).

VxWorks-vi **Tasks in VxWorks, how to check if a task is running or if it crashed, how to get stack trace**

One can check the status of a task (i.e., whether it is running) with the command **i** *task*, as in the following example.

```
--> i

  NAME        ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- -------- ------- -----
tExcTask   excTask       1efeb30   0 PEND         17ea08  1efea58       0     0
tLogTask   logTask       1efc1b8   0 PEND         17ea08  1efc0f0       0     0
tShell     shell         1ecfc00   1 READY        163390  1ecf880  3d0001     0
tRlogind   rlogind       1edb8d8   2 PEND         15f9cc  1edb518       0     0
tTelnetd   telnetd       1ed99b0   2 PEND         15f9cc  1ed9850       0     0
   ...
```

READY or PEND are okay. SUSPEND is not, but if it nevertheless happens to be so, **ti** can be used to see where the task crached.

ti gives the stack trace.

VxWorks-vii **Is there a VxWorks manual and how to find it**

There is a printed version of the VxWorks manual in the trailers. [19]

The manual [20] as well as other documentation [21] can also be found online (download, pdf format).

Advanced

VxWorks-viii **Global variable and accessibility of memory, how to dump and change a region of memory**

Under VxWorks system, all variables are global, and all memory can be accessed and changed by the user (it is not protected).

Pieces of memory can be *displayed* or *modified* by specifying the VxWorks commands **d** and **m** together with respective pointer variable to that memory region. [22]

[19] Ask Ilya, ikrav@fnal.gov
[20] http://www.windriver.com/pdf/ref.pdf
[21] http://d0server1.fnal.gov/www/online_computing/projects/controls/vxworks.html
[22] Check help on VxWorks prompt.

VxWorks-ix **How to build and run little C programs under VxWorks**

For creating an object file, one uses the *crosscompiler* for VxWorks in Linux (one of the Gateways).

As an example, the list of commands needed to compile atmtest, a module which is meant to be used to test ATM connections, as later described, is as follows

```
evbproxy@b0l3gate2> setup -q mv2603 vxworks v5_3e
evbproxy@b0l3gate2> setup cdfevb_seqio
evbproxy@b0l3gate2> ccppc $VX_MV2603_OPTS -I$CDFEVB_SEQIO_DIR/inc atm_rw.c
```

This will produce the file *atm_rw.o*, which has been renamed atmtest.

After compiling, move the .o file to the vxboot node. [23]

In order to load it, in vxworks prompt use

```
-> ld < file.o
```

Only then can it be executed,

```
->  < program
```

After loading the module, the moduleShow command should list it, among other modules. To unload the module, when no longer needed, use [24]

```
-> unld "file.o"
```

[23] E.g, to one's home directory on that node.

[24] Do not forget the double quotes.

1.4 VRB

VRB-i **What are VRBs for, where can VRB manual be found**

VRB (*VME Readout Board*) boards are multi-port memories used to buffer and filter data (pending a Level 2 trigger decision) prior to be transferred to SCPUs.

Each of the 15 VME (a.k.a. VRB or SCPU) crates [25] possesses one SCPU that controls the various VRBs in the same crate. Each VRB itself receives event data from front-end (FE) crates. A VRB contains a 10 (numerated 0 to 9) independent input ports (data channels from FE system) and a common VME output port.

A manual is available online [26] which specifies VRB details.

VRB-ii **What is the difference between the SVX and DAQ VRBs**

Buffer management for DAQ VRBs is provided by the VRB internal logic.

In the case of SVX VRBs, it is a single controller board (the Silicon Readout Controller or SRC, located in one of the Silicon crates) communicating with all SVX VRB modules that is is responsible for buffer management.

Focus on DAQ VRB. It works as a Level 3 input buffer (no data discarded, no external control necessary) operating in FIFO (*first-in-first-out*) mode. Each of the 10 channels has a buffer capacity of 64 Kbyte. Event data is organized by 4 byte -long words. [27]

VRB-iii **What are enabled and emulated channels, who sets them**

Each of the 10 channels per VRB module can be either enabled or disabled. This is done changing the database [28] accordingly.

It is also possible to emulate a particular channel (it generates itself 'data') for testing purposes.

VRB-iv **What is resetting VRBs, how and by whom it is done**

The operation of VRB resetting involves clearing all data buffers.

Preparing for the START state implies that everything is initialized properly, in a given sequence. Thus, when instructions from RC arrive – after passing through EVB Proxy, Scanner Manager – SCPU instructs VRBs to reset (through appropriate *control registers* in VRB).

The board itself can also be reset manually.

[25] These are the VME crates located on the first floor of B0.
[26] http://www-ese.fnal.gov/eseproj/svx/vrb/vrb.pdf
[27] The upper limit for resident events on memory is actually 64.
[28] **cardEditor** in online machines.

VRB-v **Who are the contact people if we have VRB problems**

If problems arise related to SVX VRBs one should contact the silicon pager. [29] If they related to DAQ VRBs there are also contact people. [30]

VRB-vi **What is the structure of raw data coming from VRB, what is VRB header**

Output data from a VRB has a specific format. Each *VRB segment* has some header information added, together with the *link fragments*.

Advanced

VRB-vii **Direct VRB access**

One can read VRB data directly, check VRB registers and test FE-to-VRB connection without using the Event Builder. There is a number of tools in CDF online software developed for that purpose. The tools evolve continuously and are mainly used by the people responsible for VRB problems. [31] These tools are sometimes useful for EVB experts as well, in which case help from DAQ VRB experts should be sought.

Figure 1: VRB board.

[29] SVX: for deeper problems the contact person is Steve Nahn.
[30] DAQ: Jim Patrick or (if hardware need to be replaced) Fred Lewis.
[31] At the moment, these people include Jim Patrick and Frank Chlebana.

1.5 Scanner CPU and Scanner Manager hardware

SCPU/SM-i **What is on-board computer**

CPU embedded on a board.

It is the case of SCPU in VRB crates (a.k.a. SCPU crates).

SCPU/SM-ii **Where is PCI interface and ATM card**

PCM ATM card directly connected to SCPU board.

ATM network connection to Converter nodes is via PCI ATM.

SCPU/SM-iii **Rules to insert CPU into adapter and adapter into crate**

Make sure power is turned off during procedure.

First insert CPU board into adapter. Then pull adapter into crate and adjust handlers.

As a general rule, before using the crate after power cycling must *clean EVB*.

SCPU/SM-iv **How CPU connects to network**

Data received by SCPU from VRBs is directed to a Converter via **ATM** network. The connections from SCPU to ATM switch (and from this to Converters) are ATM (optical) fibers. ATM is a data network.

SCPU communicates to **SCRAMNet** network, using SCRAMNet card (slot 2) through backplane, and the card connects to ring [32] through SCRAMNet bypass. SCRAMNet is a control network, managed by Scanner Manager.

SCPU connects to **Ethernet** network.

SCPU connects to **minicom** network via serial ports.

SCPU/SM-v **The transition module 712, how to connect it and check if it is functioning properly**

This module is located in the VRB crate on the side opposite to SCPU relative to backplane.

It possesses an Ethernet and several serial ports.

A serial port is used by minicom network. The minicom network cables originating from the 712 transition module are directed to one of the minicom switches (located on the 1st floor, in the rack next to the crate's - 1RR15D)

The Ethernet cable provides the connection of the SCPUs to the internal L3 Ethernet farm. Ethernet cables from the various VRB crates go to a *patch panel* (on

[32] SCRAMNet network has a ring topology.

the front side, bottom part of the rack) before they are directed to the SCPU hub [33] on the 3rd floor (lhs after entering L3 farm room).

Proper functioning is indicated by orange and green leds turned on.

SCPU/SM-vi **What is serial port, how to connect to serial port for SCPU and SM; how to use minicom, what are the keys; is it any different from Ethernet connection**

Serial ports in the case of Linux boxes are located on its back side. [34]

In the case of SCPUs, the minicom connection is done through serial port on the *transition module*, on the opposite side of the backplane with respect to the SCPU.

The case of Scanner Manager is different, and the connection to minicom must be done by connecting the serial port of the SM crate to that of b0pcom2.

Minicom is a simple network that allows to execute a few simple commands. In order to use it it is necessary to first log in to the minicom server, **b0dap10.**

```
b0dau30~ > ssh b0dap10
<b0dap10.fnal.gov> minicom b0eb11
ctrl-A Z
```

Minicom commands are introduced as *ctrl-A key* . The command *ctrl-A Z* displays the minicom *keys* meaning, as follows.

```
                Main Functions                    Other Functions

Dialing directory..D  run script (Go)....G | Clear Screen.......C
Send files.........S  Receive files......R | cOnfigure Minicom..O
comm Parameters....P  Add linefeed.......A | Suspend minicom....J
Capture on/off.....L  Hangup.............H | Exit and reset.....X
send break.........F  initialize Modem...M | Quit with no reset.Q
Terminal settings..T  run Kermit.........K | Cursor key mode....I
lineWrap on/off....W  local Echo on/off..E | Help screen........Z
                                           | scroll Back........B
```

Ethernet connections use a different port (Ethernet port).

[33] in general, a *hub* is a common connection point for devices in a network; a *switching hub* on the other hand actually reads the destination address of each packet and then forwards the packet to the correct port.

[34] Where one would connect e.g. the keyboard.

1.6 EVB-L3 Ethernet Network

Ethernet-i **What is approximate diagram of interconnections of L3 internal network** [35]

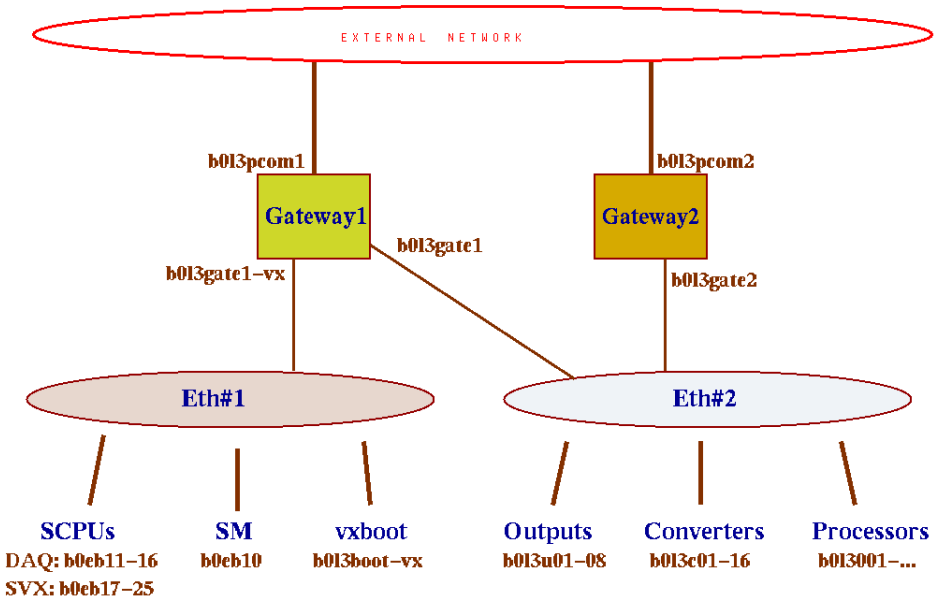


Figure 2: Internal EVB/L3 Ethernet network.

Gateway1 and Gateway2 contain the EVB and L3 proxies resp., and serve has doors connecting the internal Ethernet network to the outside world. They have both external and internal IP addresses assigned. External are 131.225.236.188 (b0l3pcom1) and 131.225.236.189 (b0l3pcom2). Internal are 192.168.24.11 (b0l3gate1, seen from l3 farm sub-net), 192.168.20.128 (b0l3gate1-vx, seen from SCPUs sub-net), and 192.168.24.12 (b0l3gate2, seen from L3 farm only).

All the other machines in the internal network (L3 farm and EVB) have internal (but not external) IP addresses (of the standard form 192.168.x.x). [36]

The L3 farm (including Converters, Processor and Output nodes) communicate to each other forming one sub-net, which also includes both Gateways and Ethernet connection to the ATM switches.

A separate sub-net incorporates the SCPUs, Scanner Manager and vxboot machine (both SCPUs and SM boot from this machine, b0l3boot-vx, located on the third floor) and only one of the Gateways, Gateway1 (see 2).

[35] Sometime in November 2001 this was slightly modified, as problems occurred after increasing the size of the network; basically it was split into two (almost independent) sub-nets, communicating through Gateway1.

[36] To be sure, one can only connect to any of these internal machines after log into the Gateways.

Both sub-nets are almost separate, Gateway1 providing the link between both. [37]

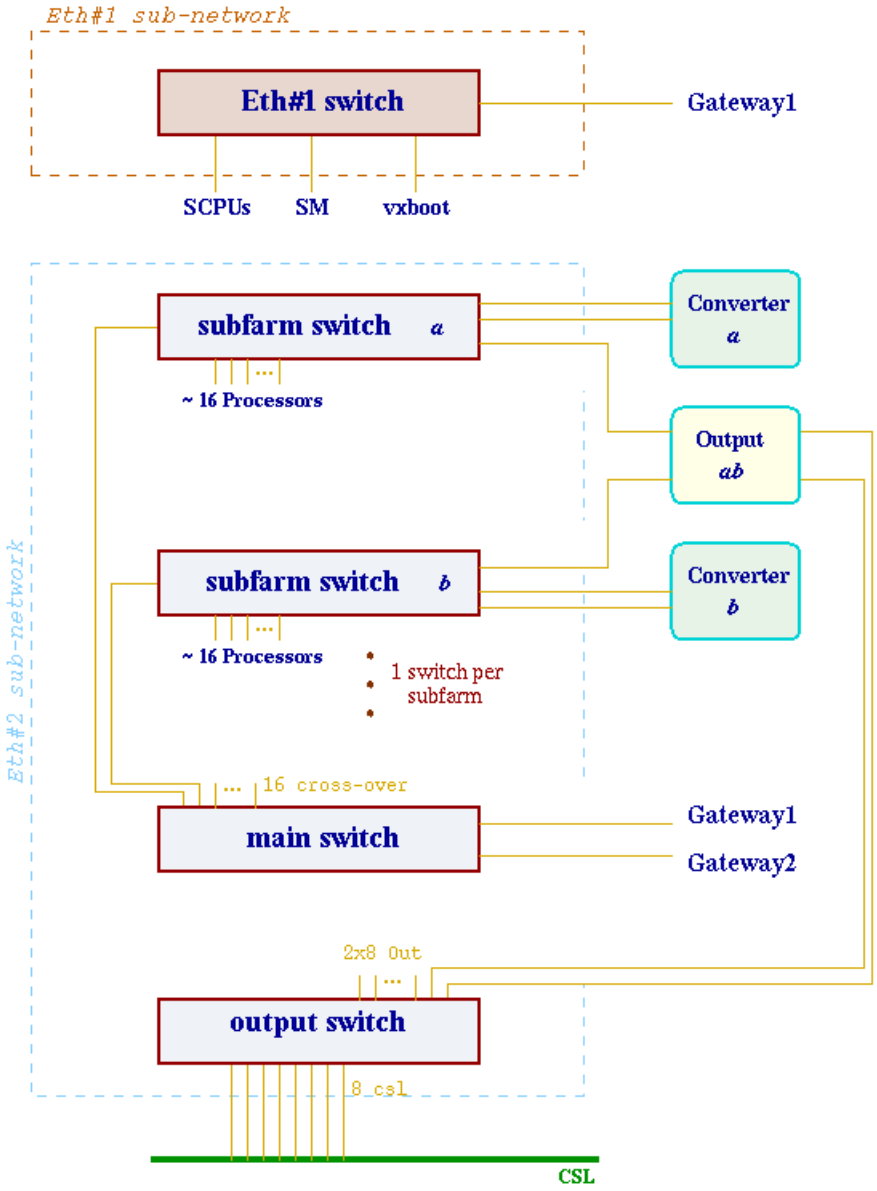


Figure 3: EVB/L3 Ethernet switches and physical connections.

Ethernet switches are located on the third floor.

The topmost one is responsible for one of the sub-nets, namely the one involving the SCPUs, SM and vxboot.

The other switches are responsible for the other sub-net, namely the one including the L3 farm. 16 of these are associated with the sub-farms, and get connected

[37] E.g. the Converter nodes need to communicate to Scanner Manager, and these are not in the same sub-net.

together by cross over cables going from each one to the *main switch*. An additional switch is associated with the Output nodes, and provides connection to CSL (see 3).

The topmost switch has 4 cable connections respectively to Gateway1, SM, SCPUs (which cables are bundled together by an extra hub also in the third floor) and vxboot.

Each of the switches of the L3 farm subnet are assigned to each one of the 16 sub-farms. In particular each has 2 connections (read *cables*) to the sub-farm Converter, [38] 16 to the 16 [39] Processor nodes in the sub-farm, 1 to the Output node, and 1 cross over cable to the main switch).

The main switch receives 16 crossover cables, and one cable from each Gateway; it also receives an additional cable providing Ethernet connection to the ATM switches (b0atm1, b0atm2).

The output switch receives 2x8 cable connections from the 8 Output nodes, [40] and 8 connections to CSL (*Consumer Server Logger*). [41]

Ethernet-ii **How are we connected with online cluster and CSL computers**

Connection of the internal network to the online cluster is done via the two Gateways.

The output Ethernet switch connects directly to CSL.

Ethernet-iii **What is the Ethernet connection between SCPUs and the 3^{rd} floor**

The SCPUs connect to one of the internal sub-nets (see above), namely through an extra hub (also located on the third floor) that is connected to the 'topmost' switch in the rack.

Ethernet-iv **How to see if a switch is okay and how to reboot it (lights, etc)**

Can check leds — to each connection there corresponds a pair of lights; normally they should be one steady green (OK) and one flashing orange (data).

Rebooting is done by power cycling the switch. [42]

Advanced

[38] The reason there are two instead of a single cable, is that due to bandwidth reasons each is assigned half of the sub-farm.

[39] Actually currently the number of Processor nodes per sub-farm is not uniform, it can be 18, 16 or 14.

[40] Each Output node has 4 cable cable connections – it receives data from 2 sub-farms, and 2 cables connect to the output switch.

[41] CSL is b0dau32; and CSL backup is b0dau31.

[42] No need to put nodes offline.

Ethernet-v **How to test Ethernet connection**

Can try to *ping* and *ssh* to a particular node.

The entire farm can also be ping-ed by executing the script **$L3_CSHDIR/l3_ping_farm**, or by conveniently making use of **$L3_CSHDIR/l3_all_nodes_exe**. [43]

[43] Executed from b0l3pcom2 (as user l3proxy).

1.7 ATM switch

ATM-i **What is ATM**

ATM (Asynchronous Transfer Mode [44]) is a (widely employed in telecommunication systems) mode for transferring data over networks.

Some general facts about ATM:

It uses short, fixed-length packets called cells for transport; information is divided among these cells, transmitted and then re-assembled at their final destination.

It provides a single network for all traffic types-voice, data, video. Allows for the integration of networks improving efficiency and manageability.

It is not based on a specific type of physical transport, it is compatible with currently deployed physical networks. ATM can be transported over twisted pair, coax and fiber optics.

In the Event Builder System at CDF, an ATM network is used to transfer data from SCPUs to the L3 farm.

ATM-ii **What hardware components it has**

The network uses the *AAL5* protocol, that allows for proper 'hand-shaking' of involved hardware components. These include the **ATM switch**, the **PMC ATM card** on the SCPUs (and SM), and the **PCI ATM module** on the Converter nodes. All these are connected together through (pairs of) optical connections. [45]

The ATM switch itself is formed by an input part, a *switching fabric*, and an output part. Both input and output parts contain 1 CPU, *b0atm1 and b0atm2,* resp., and 4 network modules. Each of these modules has 4 connections, each formed by receiver and transmitter plugs.

ATM-iii **What are VPI and VCI**

The complete address (VPI,VCI) specifying the transport of ATM cells is done using **VPI** (*Virtual Path Identifiers*) and **VCI** (*Virtual Channel Identifiers*).

While a **VCI** describes a unidirectional transport of ATM cells, the **VPI** identifies a unidirectional transport of ATM cells belonging to a bundle of virtual channels.

ATM-iv **How to test ATM connection between SCPUs and Converters**

One can log into both ATM CPU and SCPU and execute existing programs created for the purpose. Consider the following ('synchronized') sequence of instructions on both a given SCPU *and* a Converter node.

[44] Not Automated Teller Machine ...

[45] An optical connection maximum data transfer rate is $\sim 16.2\ Mbps$.

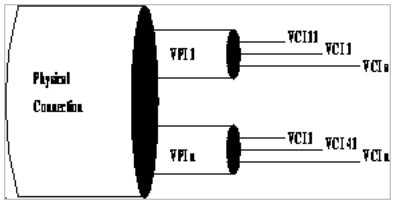


Figure 4: VPI and VCI

```
                                                root@b0l3c02> /root/atm/br 1.1

 evbproxy@b0l3gate1> ssh b0eb11
 -> ld <  ~vxboot/ikrav/atmtest
 value = 21434672 = 0x1471130
 -> atm_write(2,1)
 nwrite = 13
 value = 0 = 0x0
                                                Hello, world
-> logout
evbproxy@b0l3gate1>
                                                Ctrl-C
                                                root@b0l3c02> exit
                                                b0l3c02> exit
```

Log in as *evbproxy* on the SCPU and as *root* on the Converter.

On the Linux machine (Converter node) run **/root/atm/br** with argument list *VPI.VCI*, where *VPI* denotes the number (see bellow) of the SCPU one is using for the test (and just log in) and *VCI* is a number (0,1,2). [46]

On the VxWorks machine (SCPU), after loading (**ld**) the atm module into memory, one runs **atm_write(***VPI'***,***VCI***)**, where *VPI'* is the number (see below) of the Converter one is using and *VCI* is the same number used above. One would then expect to see the string " *Hello, World* " printed on the Converter's console.

The check of the connection between SCPUs and converters can be performed automatically using the Expert's control in the Ace Control Panel (from version 2.20). The procedure described before is implemented in an expect script, which takes care of loading/unloading the atm module in the selected SCPU, starting/killing br on the converter and sending 1000 4-characters words, checking that all of them are received by the converter. The script exits printing a summary withthe number of words that have been received. The GUI allows the user to specify the SCPU, converter and port number.

Note: The *number* said above to be associated with the nodes is determined as follows: in the case of the Converters, this *number* coincides with its CPU name termination (i.e.,

[46] A certain set of VCI numbers are reserved for system use.

b0l3c#), so *number*=#; in the case of SCPUs it corresponds to its CPU name termination (i.e., b0eb#) after subtracting 10, so *number*=#-10. This correspondence can be checked explicitly at the bottom of the */etc/hosts* file on the Converters, where the ATM addresses are such that VPI is in the address' last position.

ATM-v **How to check hardware on SCPU, ATM and Converters looking at LEDs**

On Converters one can look at file /cron/atm/devices.

```
b0l3c01> more /cron/atm/devices
Itf Type    ESI/"MAC"addr AAL(TX,err,RX,err,drop) ...
0 nicstar 001c40000f48  0 ( 0 0 0 0 0 )  5 ( 0 0 29263817 0 1 )
```

One can recognize the protocol designation AAL5. Looking at the last brackets, one identifies the correspondence between the number of transmitted (Tx) and received (Rx) 'messages', and respective errors (err), and number of dropped messages. One sees as expected zero on the column for transmitted, and a great number for received, as it should be for a Converter node (relatively to the ATM network).

Advanced

ATM-vi **How to understand problems by logging in to ATM CPU**

One can log in directly to the ATM CPUs (b0atm1, b0atm2) and perform a number of informative commands, hoping that will give hints towards understanding a problem.

```
b0l3gate1> telnet b0atm1
login:
b0atm1::> help

 General commands:
'?' to get list of commands at the current level
'up' to go up one menu level
'top' to go to the root menu
'exit' to leave AMI

about                  - Display program information
close                  - Close this connection
configuration>         - System configuration submenu
debug>                 - Switch debug submenu
display>               - Switch display submenu
exit                   - Exit AMI
help                   - Display help for each command
```

```
history                 - Display command history
open                    - Open a connection
operation>              - Switch operation submenu
ping                    - Ping a host or switch
redo                    - Repeat a history command
rows                    - Get/set number of rows
startup                 - Switch configuration wizard
statistics>             - Switch statistics submenu
top                     - Go to the root menu
up                      - Go up 1 menu level

b0atm1::> statistics
b0atm1::statistics> help

General commands:
'?' to get list of commands at the current level
'up' to go up one menu level
'top' to go to the root menu
'exit' to leave AMI

atm>                    - ATM layer statistics submenu
atmroute>               - Atm route statistics submenu
cec>                    - Common Equipment Card submenu
cese1                   - Display CES port statistics
cesds1                  - Display CES port statistics
ces                     - Display CES connection statistics
cr                      - Display callrecord statistics
board>                  - Board statistics submenu
fratm>                  - FR/ATM statistics submenu
funi>                   - FUNI statistics submenu
ipaccess                - Display IP filtering statistics
iwf>                    - Interworking Submenu
module>                 - Display Netmod statistics submenu
nsapfilter>             - Address Filtering Statistics submenu
oam>                    - OAM statistics submenu
port                    - Display general port counters
reset                   - Reset counter statistics
portcard                - Display portcard statistics
scp>                    - Switch Control Processor statistics submenu
spans                   - Display SPANS signalling statistics
signalling              - Display signalling statistics
vcc                     - Display virtual channel statistics
vpc                     - Display virtual path statistics
vpt                     - Display virtual path terminator statistics

b0atm1::statistics> port

     Input            Output               Cells       Cells
```

```
Port  VPs   VCs     BW  VPs   VCs     BW    Received Transmitted ErrSecs Ovrflws
1A1    16     0   0.0K    0     0   0.0K   776512820           0       0       0
1A2    16     0   0.0K    0     0   0.0K  1280241857           0       0       0
             ...
b0atm1::statistics> exit
b0l3gate1>
```

EVENT BUILDER

2 Event Builder

2.1 EVB dataflow mechanics

The 'mechanics' of EVB data flow is described with detail in a separate document.[47] The diagram below contains the actions involved in starting a run, loading and sending an event. [48]

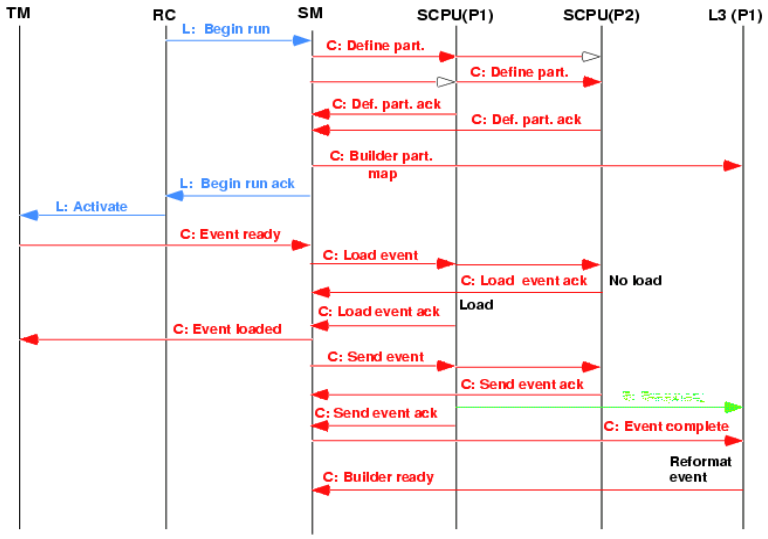


Figure 5: Messaging of single event.

PROCESSING 1 EVENT
The following sequence of messages are involved in the processing of a single event.

1. TM → *SM* *new event* (whenever L2 accept)
2. SM → SCPUs *load event*
3. SCPUs → SM *acknowledge: load event*
4. SM → SCPUs *send event* (to available builders)
5. SCPUs → SM *acknowledge: sent event*
6. SM → Boss x (of a unique Converter) *event complete*
7. Boss x → SM *acknowledge: received event*

[47] S.Tether, *Event builder messages*, ~leonardo/Public/l3evb/docs/EVBmessages.ps

[48] The participants – Trigger Manager, Run Control, Scanner Manager, SCPUs and Converter node – are represented as vertical lines; time evolves from top to bottom; **C:** denotes control network messages and **L:** refers to Ethernet (local area network) communications.

8. SM → TM *done*

There is a limit of 7 simultaneous events in the EVB; the *done* messages are necessary for the TM to keep track of the number of events currently being processed.

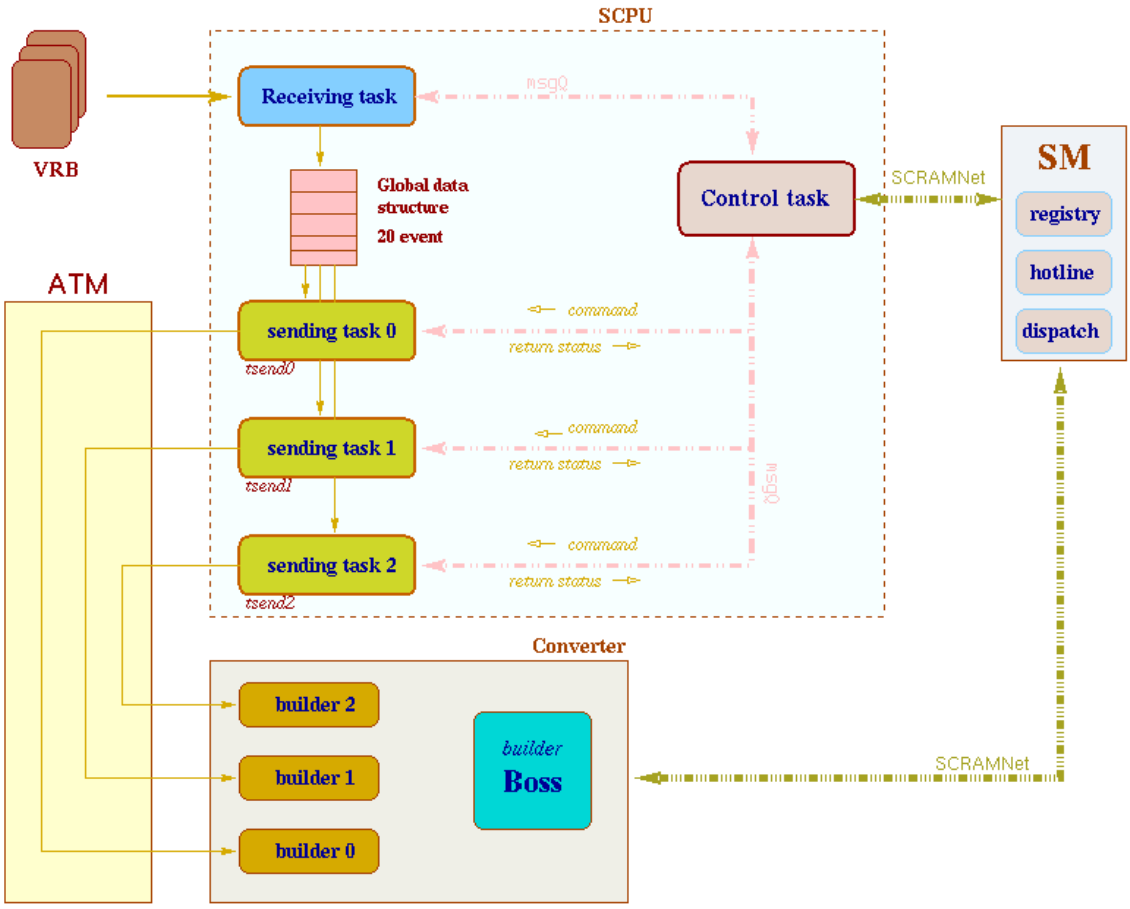


Figure 6: Communications of EVB processes.

PROCESS COMMUNICATION

Various processes on the SCPUs, Converter nodes and Scanner Manager are responsible for assuring the proper sequence of actions leading to the processing of events by EVB.

On the SCPUs there are *receiving, sending* and *control* tasks. *Receiving* obtains data from VRBs, and transfers it when appropriate to *sending* through a global data structure, that works additionally as a 20 event buffer. *Control* communicates with SM using SCRAMNet memory, and communicates SM control commands to *receiving* and *sending* using message queues. While there is a single *receiving* (tRec_0) and a single *control* (tCtrl_0) processes running on each SCPU, there are various *sending* (tSend_0 up to tSend_47)

tasks, three per Converter to which it communicates. [49]

On the Converter nodes, *l3_node* contains *builders* and a *builder boss* threads. Each builder receives data from the corresponding *sending* tasks (one from each SCPU) through the ATM network. The *builders* do not communicate with the SM directly; instead, the *builder boss* sends and receives all SM messages on their behalf, using appropriate SCRAMNet memory regions.

The Scanner Manager contains three tasks: *registry* (tsmReg), *hotline*(tsmHot), *dispatcher* (tsmDisp), which should be running. [50]

[49]This is checked explicitly simply by logging on an SCPU and list the running tasks; e.g., **telnet b0eb**12 ; **i** ; **logout** (for explanation of VxWorks commands refer to a previous section).

[50]This is checked explicitly simply by logging in to the SM and list the running tasks: **telnet b0eb**10 ; **i** ; **logout**

2.2 SCRAMNet

SCRAMNet-i **What is SCRAMNet for**

SCRAMNet (Shared Common Random Access Memory Network) is a real-time applications network. It is used as a control network among the EVB components. It connects Scanner Manager, Trigger Supervisor, SCPUs (through VMS SCRAMNet module) and Converter nodes (through SCRAMNet PCI card) in a serial-ring network.

The ring sequence is the following:

$$\ldots \rightarrow SM \rightarrow b0l3c01 \rightarrow \ldots \rightarrow b0l3c16 \rightarrow b0eb25 \rightarrow b0eb24 \rightarrow$$
$$\ldots (even\ SCPU\ \#s) \ldots \rightarrow b0eb12 \rightarrow b0eb11 \rightarrow \ldots (odd\ SCPU\ \#s) \ldots \rightarrow$$
$$b0eb23 \rightarrow b0tsi00 \rightarrow SM \rightarrow \ldots$$

Each processor on the network has access to its own local copy of shared memory, which is rapidly updated.

It is a replicated shared memory — any data written into memory is automatically sent to the same shared memory location in all nodes on the network.

It uses paired-fiber-optic transmission media.

Some facts about SCRAMNet:

- the SCRAMNet cards in each node communicate over a serial ring architecture
- 150-megabit/second fiber optic communication system
- network transfer rate up to 16.7 MB/s
- A datum is communicated when a CPU makes a high-level language statement: A=B, where the variable A is located in the shared-memory window of the SCRAMNet card. Within microseconds all other CPUs on the network have the same variable A in their shared-memory areas. The SCRAMNet card does this immediately and automatically by passing both the datum and the datum address over the network. The variable is written to the same relative shared-memory address in each computer's shared-memory area by the receiving SCRAMNet cards.
- data transfer is handled at the hardware level, no software required
- all nodes have equal priority for network bandwidth.
- 256 node capacity in a ring

SCRAMNet-ii **SCRAMNet card for VxWorks, for PC**

When a host node writes to the shared memory, the proper handshaking logic is supplied by SCRAMNet host adapters.

Each SCPU connects via backplane to a VME6U module that occupies one slot on the VME bus chassis, and this module connects directly to a bypass switch and to the SCRAMNet network.

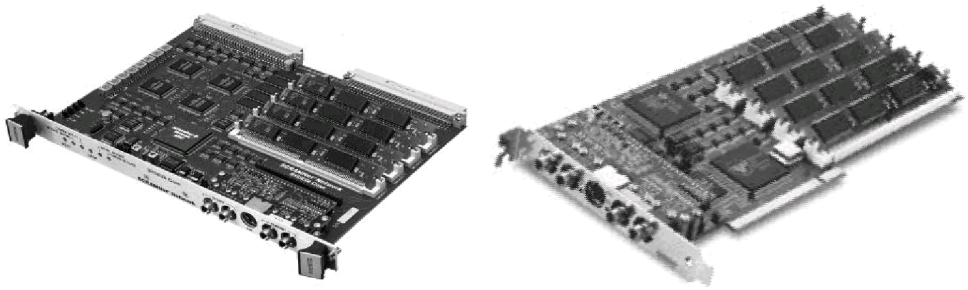


Figure 7: SCRAMNet cards: (i) VME6U module and (ii) PCI card.

Converter nodes communicate to the SCRAMNet network through a PCI card.

Both boards have a pair of transmitter (Tx), a pair of receiver (Rx) optical connections, and a power connection.

SCRAMNet-iii **Bypass switch**

A ring configuration network depends on connectivity – in case any one node in the network would not be able to retransmit incoming messages, the ring would be broken.

A bypass switch, included together with each node, automatically provides an alternative route for the optical message to travel. In case any node breaks (e.g. is un-powered) or is chosen not to be included in the ring, it is bypassed.

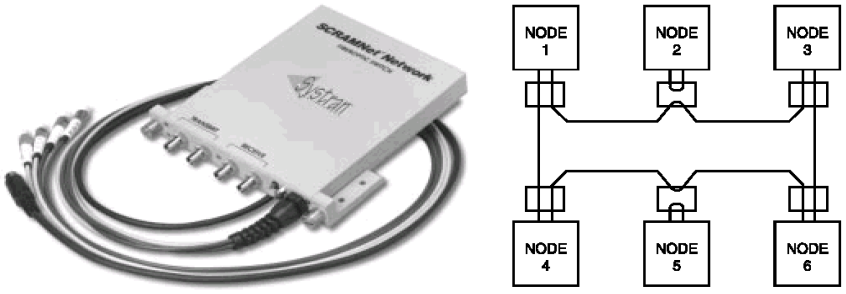


Figure 8: SCRAMNet bypass switch (i) assures serial ring (ii) continuation.

This way it increases the system's fault tolerance and flexibility.

SCRAMNet-iv **Which LED indicators must/must not be lit in normal mode**

Sets of LED indicators on both types of SCRAMNet boards may be useful in detecting an eventual malfunction.

The VME modules have the following indicators:

- **insert** indicates proper insertion of node into the network (green) [must be on]
- **message waiting** (green)
- **carrier detect** indicates a valid pair of transmit lights from the previous node into this node's receiver pair (green) [must be on]
- **error** (yellow)
- **native message** lights when message received was originated by the node
- **foreign message** lights when message received is from another node

The PCI cards have the following indicators:

- **insert** indicates proper insertion of node into the network (green) [must be on]
- **carrier detect** indicates a valid pair of transmit lights from the previous node into this node's receiver pair (green) [must be on]

SCRAMNet-v **How EVB uses SCRAMNet, SCRAMNet memory, areas reserved for computer-computer communications, how to run and understand output of dumpEvb tool**

The SCRAMNet (RAM) memory is divided into 51 regions. Each node writes to a particular region. The communication of each node to SM is given by writing and reading messages that are sequentially written to the specified memory regions.

The 51 memory regions are reserved [51] for the following communications.

- SM uses one region to broadcast to all SCPUs (0)
- Each SCPU uses its own region to talk to the SM (1-16)
- Each Builder Boss (on Converter nodes) uses one region to send to the SM (17-32)
- SM uses one region to send to each Builder Boss (33-48)
- Scanner Manager to Trigger Manager (49)
- Trigger Manager to Scanner Manager (50)

In parenthesis are indicated the region sequence identifiers. These can be given as argument to the **dumpEvb** tool, [52] which is useful to check the event processing status in the specified component. For example,

[51] This can be checked explicitly by looking at the header file $CDFEVB_SM_DIR/inc/evbScram.h in b0l3pcom1 (need to previously setup cdfevb_sm).

[52] Which can be used on any SCPU.

```
[evbproxy@b0l3gate1 ~]  telnet  b0eb12
-> dumpEvb(50)

********** TM->SM scramnet messages *********
address    slot  Word_1   Word_4  break  part.  event    command

0xd3019000  0  10002685  00000002          0     9861   Event ready
0xd3019018  1  10002686  00004002          0     9862   Event ready
...
0xd30197e0 84  10002684  00000002          0     9860   Event ready

value = 1 = 0x1
->  logout
```

Each message is composed of 6 words (1 word = 32 bit). Each of the memory regions has a size of 2 Kb, and a capacity of 512 words. Each can then accumulate 85 messages, as shown in the example above.

Advanced

SCRAMNet-vi **How to explicitly test SCRAMNet communications from Linux and VxWorks**

In VxWorks, the (global) variable scramnet_ram points to the beginning of SCRAMNet memory. The commands **d** and **m** can then be used do respectively *display* and *modify* a given memory slot, specified using the scramnet_ram pointer. [53]

To test where the SCRAMNet communications start to fail, one could login in a given SCPU, and use the command **m** to propagate a given memory modification, and by logging on directly to other SCPUs in the ring sequence try to find (using the command **m**) where the propagation ceases to occur.

SCRAMNet-vii **Full understanding of the EVB messages**

As mentioned above, a command is a sequence of 6 words each 32 bit long. A variety of information is encoded in the command via bit patterns, such as event ID, partition number, action to be taken, etc. In most of the cases, Scanner Manager issues commands and collects acknowledgments from SCPUs and L3 converters, where acknowledgments are just repetitions of the original command.

One can generally figure out how to decode SCRAMNet commands by reading the source code. The bit patterns and word contents are defined in the file $\$CDFEVB\_SM\_DIR/inc/evb$ and the SCRAMNet regions for communications between all components are defined

[53] An example of this, after login into an SCPU, could be **d** scramnet_ram,4,512 ; or something like **d** scramnet_ram+2048.50 .

in $\$CDFEVB\_SM\_DIR/inc/evbScram.h$; one has to run (as evbproxy@gate1) *setupcdfevb_sm* for the variable $\$CDFEVB\_SM\_DIR$ to be defined.

Of course, a convenient way to learn about the SCRAMNet command contents is to talk to the primary EVB expert. [54]

[54] At this time, Steve Tether.

2.3 Hardware database for EVB

Hardware dbase -i **What is hardware database, how to connect to it using cardEditor**

The CDF Hardware database is an ORACLE database that contains full information about all DAQ electronics. [55] It includes e.g. which SCPUs are currently marked online, which VRBs belong to which crates and again which are marked online.

The gui application allowing to check and edit (mark components online/offline) database configuration is the *hardware database selector*, and can be displayed from the online machines as follows

setup fer
cardEditor

The Ace Control panel too has a feature that allows to change the online/offline status of a L3 PC. The *Expert's Hall* has a button which allows to get controls for the ORACLE database. The GUI represents each L3 PC by a check button, which is checked if the PC is marked online. The user can change the status of the PCs and commit the changes made to the database, if in possess of a database account with sufficient privileges. When the GUI is properly closed, a log file is written in /cdf/log/cdfevb/. It contains all the operations performed during the database updating session, and whether the changes have been committed or not.

Hardware dbase -ii **What are all the tables: crates, tracers, racks, etc.**

The database is the cross- referenced set of tables. Its structure resembles the physical layout: racks contain crates, these contain cards (CPUs, VRBs, tracers, etc.).

Hardware dbase -iii **What is online flag for a component, what does it matter if it is on or off, by whom it is used**

One can mark a given component in the database on or off by selecting the component in cardEditor, choose Edit/View from the Edit menu, and change the value of the ONLINEFLAG to 0 (off) or 1 (on). When EVB/L3 proxies are started, this is checked. If a component is marked offline (0) it will not be used in the subsequent run.

Hardware dbase -iv **Where to find entries for SCPUs**

In cardEditor, descend into appropriate racks to find SCPUs – i.e. DAQ_VRB_00 or SVX_VRB_03.

[55] The connection happens during "Partition" transition.

Hardware dbase -v **How to find out which VRB is located in which EVB crate and what FE crates are connected to that VRB**

From online machines, **hdump** allows to find connections between tables.

setup fer
hdump

Then, click on buttons correspondent to VRBtoTracers or VRBtoFIBs.

Hardware dbase -vi **How to change database entry**

In order to change a database entry one needs to have an oracle db account.

In cardEditor need to *change database connection* and only then select the component and edit the intended entry.

2.4 EVB proxy account

Proxy account-i **How to log in for yourself and for the Aces**

Login into evbproxy account

ssh -l evbproxy b0l3pcom1

Logging on to the Gateway is allowed only from the CDF online cluster, and require valid kerberos tickets.

Access to the account is restricted, and in case one is not allowed to login directly, it may be necessary to login firstly to b0dap57 as cdfdaq and from there get the proper kerberos credentials. [56]

Proxy account-ii **.cshrc file, what environment has to be set**

The account's environment is set in .cshrc file. Setups include setting up ups products, both local and from the mounted online ups directory, [57] and setting up various environment variables. [58]

Proxy account-iii **How to run ace control panel**

In order to run the ace control panel for EVB and L3, one needs to login to b0l3pcom1 and run **ace.**

For example (using one of the computers in the Control Room)

```
<b0dap57.fnal.gov>  xhost + b0l3pcom1
<b0dap57.fnal.gov>  ssh b0l3pcom1 -l evbproxy
[evbproxy@b0l3gate1 ~]  setenv DISPLAY b0dap57:0.0
[evbproxy@b0l3gate1 ~]  ace
```

Proxy account-iv **What is zephyr, how to run zephyr window, what it's used for**

Zephyr is a message transport and delivery system — it is a way for users and machines to send real-time (nearly instantaneous) messages to each other.

The primary process that Zephyr uses is **zwgc** (the main zephyr client), or the Zephyr WindowGram Client, which is responsible for displaying zephyr notices.

The zephyr window is run by typing

[56] To do that, need to type: setup kerberos; /usr/krb5/bin/kinit -k -t /var/adm/krb5/cdfdaq.keytab; cdfdaq/cdf/'hostname'

[57] Including: hardware database (hdwdb), zephyr, cdfevb_ace, cdfevb_proxy, merlin, oracle, fer, java.

[58] As Java's CLASSPATH, ...

```
[evbproxy@b0l3gate1 ~] green
```

The zephyr messaging system listens to informational, warning and error messages from SM, SCPUs to whomever cares to listen (i.e., *subscribes*). Those error messages that are most important will be forwarded to CDF Error Logger automatically (by ZMConverter).

The *green* window is used for more in-depth problem debugging.

2.5 EVB proxy process

Proxy process-i **Where it is run**

EVB proxy runs in b0l3pcom1.

Proxy process-ii **What are necessary supporting processes**

In addition to *EvbProxy*, which is responsible for Run Control – EVB communications in general, some processes need to be permanently running: *zwgc* (Zephyr WindowGram Client) and *ZMConverter* (Zephyr-Merlin converter), which [59] are responsible for transmission of error messages from SCPUs and SM to Error Logger and Run Control.

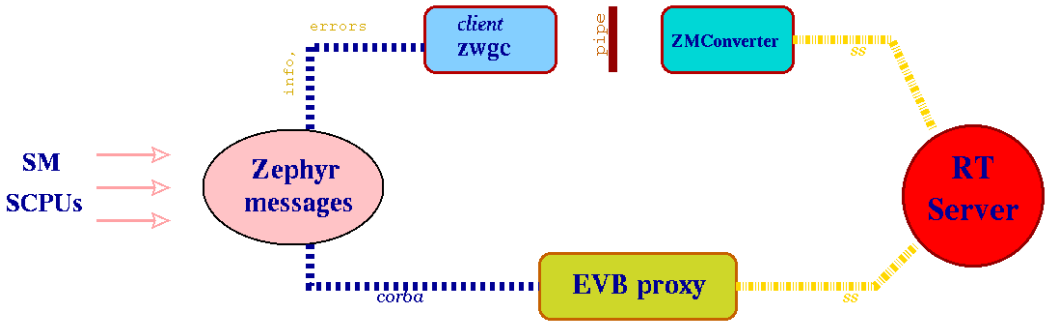


Figure 9: EVB monitoring processes.

Proxy process-iii **How to check if it is alive**

The easiest way to check this is to use the corresponding button in the Ace Control Panel.

Proxy process-iv **Where are the log files**

The various processes referred to above have their correspondent log files.

- *EvbProxy* b0l3pcom1:/cdf/log/cdfevb/evbproxy-....log
- *zwgc* b0l3pcom1:/cdf/evbproxy/proxy.zwgc.log
- *ZMConverter* b0l3pcom1:/cdf/evbproxy/proxy.converter.log

Additionally, whenever 'cleanup EVB' is performed, a log file is created

- b0l3pcom1:/cdf/log/cdfevb/cleanup-....log

with the information entered as the reason for the action.

[59] These are actually piped together, *zwgc* ... | *ZMConverter* .

LEVEL 3

3 Level 3

3.1 Level3 dataflow mechanics

Dataflow-i **Converters, Processors and Outputs**

For a given event, the various data pieces put together by the Event Builder system are delivered to a unique Converter node. The Converter passes the data then to one available Processor node in its respective subfarm. On the Processor, data are *reformatted* and *filtered*. In case the event passes successfully both reformatter checks and filter requirements, it is passed on to the Output node associated with the subfarm to which the node belongs to (each Output node is associated with a pair of subfarms). From there it is transferred to the CSL, for further monitoring and storage.

Dataflow-ii **l3_node: input, analysis chains and output**

Data flow at the Level 3 farm is managed by the **l3_node** executable. It exists in every single node of the farm (Converters, Processors, and Outputs), and has the same general structure. It is sequentially composed of *input*, *analysis* and *output* modules.

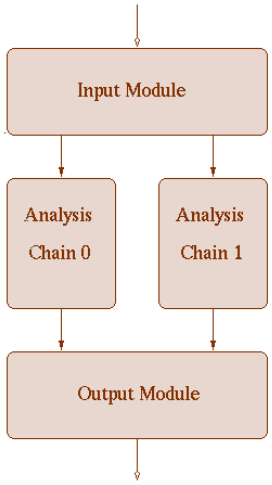


Figure 10: Structure of `l3_node`.

Input Module

The following categories exist of the input module:

1. **random** input
   It generates random data.

2. **file** input
   Gets data from a file.
3. **network** input (Converter nodes)
   Obtains event from ATM network (15 ATM connections from SCPUs). [60]
4. **faucet** input (Processor nodes)
5. **sink** input (Output nodes)

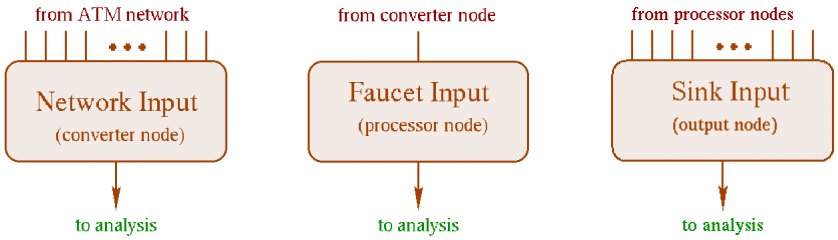


Figure 11: Input modules used in normal data taking.

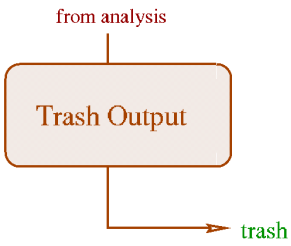
Analysis Chains

In the case of Converter and Output nodes, this is empty.

In the Processors, it is composed of two parallel chains (chain_0, chain_1) [61] each containing reformatter, and filter interface.

Output Module

1. **trash** output
   Event is discarded.



[60] All other following module connections are via Ethernet.
[61] Which are distributed by Linux – it's not imposed that each should run on which CPU of the Processor node.

2. **file** output
3. **faucet** output (Converter nodes)
   One connection towards each Processor node in the associated subfarm.
4. **sink** output (Processor nodes)
5. **drain** output (Output nodes)
   Connection to CSL.

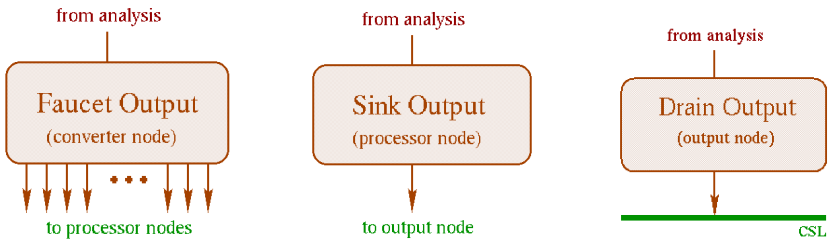


Figure 12: Output modules used in normal data taking.

Dataflow-iii **How all Level3 nodes connect to each other** In the beginning of each run the l3_node executables are started on all nodes and creating a chain of dataflow through Level3. The sequence of startup and connections is important. Before starting Level3, the two adjacent system have to be running and ready, namely the Event Builder and the CSL.

The general scheme is starting components outside-in. After EVB/CSL are ready, the l3_node executables are started on Converters and Outputs. Converter nodes establish ATM connections and exchange initialization messages with the Event Builder. Output nodes connect to the CSL. At this time Converters do not know where to send data and Output nodes do not know from where to receive data.

Finally, Processor nodes are started. From their command line options Processors know which Converters and Outputs to connect to. Indeed, Converter and Output nodes are presented as *servers* to the Processor nodes, and receive requests from these, which themselves behave as *clients*.

At the end we have a continuous connection from EVB to CSL. The procedure at the end of each run is reversed: shut down Processor nodes, then Converters and Outputs, and finally the Event Builder and the CSL.

Dataflow-iv **l3_node command line options**

The behavior of the L3 data flow executable l3_node is controlled by a number of command line arguments. First, the parameters define which input and output modules will be used by l3_node, therefore defining whether the node will behave as a

Converter, Processor or an Output. Parameters for Processor nodes are most numerous and include specifications for L3 filter executables, reformatter constants/flags, etc that come to level3 from Run Control for each run. Parameters common to all types of nodes include the partition number, timestamp, event size limit, whether to connect to monitoring system and so forth. Examples of l3_node command lines can be seen in l3proxy log files. One can learn about all available options from looking at the docs in the beginning of $L3_SRCDIR/programs/l3_node.cc of level3 package.

Several examples are also given below.

```
Command line to run l3_node as a Converter:
 Input-Network, output-Trash:
 $L3_BINDIR/l3_node --partition=7 --size=1048576 --input-network=b0eb10 --host=b0l3c01a-a
 --builders=3 --output-trash --null-reformatter >&
 $L3_LOGDIR/l3_node/l3_node.out_0005181310 &

 Input-Network, output->Processors:
 $L3_BINDIR/l3_node --partition=7 --size=1048576 --input-network=b0eb10 --host=b0l3c01a-a
 --builders=3 --output-faucet --null-reformatter >&
 $L3_LOGDIR/l3_node/l3_node.out_0005181311 &

Command line to run l3_node as a Processor:
 Input-Faucet, output->Output Node
 $L3_BINDIR/l3_node --partition=6 --size=1048576 --input-faucet=b0l3c01a:0 --builders=4
 --output-sink=b0l3u04a --converter-reformatter
 --filter=/home/ikrav/level3_development1/level3/control/bin/m486-unknown-linux2/l3_fake_filter:
/home/ikrav/level3_development1/level3/control/node_io/command_input.tcl
 --run-number=1 --run-type=1 --expt-type=9 >&
 $L3_LOGDIR/l3_node/l3_node.out_0005171530 &

 Same but using the new filter
 $L3_BINDIR/l3_node --partition=6 --size=1048576 --input-faucet=b0l3c01a:0 --builders=4
 --output-sink=b0l3u04a --converter-reformatter
 --filter=$L3_FILTER_DIR/L3_test.csh:$L3_FILTER_DIR/L3_test.tcl --run-number=1
 --run-type=1 --expt-type=9 --force-stream=1 >&
 $L3_LOGDIR/l3_node/l3_node.out_0006081036 &

Command line to run l3_node as an Output:
 $L3_BINDIR/l3_node --partition=6 --size=1048576 --input-sink --host=a --host=b
 --builders=4 --output-trash --null-reformatter >&
 $L3_LOGDIR/l3_node/l3_node.out_0006081030 &
```

Moderately Advanced

Dataflow-v **How to run a simple l3_node test on a single node (random data→trash)**

An appropriate command line for l3_node can allow one to run l3_node by itself, making it generate data at the input and discard at the output. [62] Here is such an example:

[62] This is sometimes useful when one wants to test a processor node before including it into the system.

```
$L3_BINDIR/l3_node --partition=7 --size=1048576
                  --input-random --fragment-size=16536
                  --host=b0l3c01a-a --builders=3
                  --output-trash --null-reformatter
```

In order to end l3_node gracefully, and clean up, one executes:

```
$L3_CSHDIR/l3_end_node_control
```

Dataflow-vi **What are circular buffers, how to print them with l3_debug_dump command**

Circular buffers are useful for additional dataflow (debugging) information, not stored in log files. They are fixed size (e.g. $\sim 20Kb$) memory buffers that keeps on being 'circularly' written to. Circular buffers can be printed with

```
$L3_BINDIR/l3_dump_circular <id>
```

There are also *cstate* and *xstate* arrays containing respectively dataflow and filter status information. These can be dumped with

```
$L3_BINDIR/l3_dump_cstate
$L3_BINDIR/l3_dump_xstate
```

Circular buffers, shared memory, process status, can all be printed and saved (on an internal node) using the following script

```
cd /log/l3_node
$L3_CSHDIR/l3_debug_dump l3_node.out_#  > myfile.log
```

which runs itself ipcs, *cstate* and *xstate* dumps as above.

3.2 Reformatter

Reformatter-i **What it is for**

The role of the reformatter is twofold. It performs several checks for data integrity. If some of these fail, a *reformatter error* is generated and the event is discarded right away. [63] Additionally, the reformatter rearranges the event pieces, modifying the raw event format (mini banks), which becomes now more *'physics-like'* (TRYBOS format). [64]

The reformatter package is named RAWREF (CVS/UPS). It has no independent executable; it is constituted by a set of libraries that are linked to l3_node.

Reformatter-ii **Where reformatter errors appear (l3 logs, error logger)**

Among the various reformatter checks [65] are, for example, the length of a fragment, at each level, the event ID, predefined bit patterns, and so on. There are correspondingly ~ 40 types of possible errors.

Every reformatter instance (two per Processor node) saves one event with an error of each kind . These error events are saved in each Processor node as

```
/cdf/log/l3_node/error_event_x_yy
```

where $x \in \{0, 1\}$ denotes the chain, and yy the error type. Below is the correspondence between the types and codes of the various errors.

Error Codes		Error Codes	
I-001	C-MINI-NTOTL	I-021	C-SCPUMAXVRB
I-002	C-MINI-INSTA	I-022	C-EVT-MXSCPU
I-003	C-MINI-MXBUF	I-023	C-EVT-2-LONG
I-004	C-MINI-NEXT?	I-024	C-EVTINCEVTN
I-005	C-MINI-NAME?	I-025	C-EVT-NOTFRD
I-006	C-MINI-TYPE?	I-026	C-BNK-2-LONG
I-007	C-LINKMXMINI	I-027	C-BNKNOTEVEN
I-008	C-LINK2-LONG	I-028	C-BNKNIMTYPE
I-009	C-LINKINCWCO	I-029	C-BNKNAMEINC
I-010	C-LINKSIL2MI	I-030	C-BNK-NTOTAL
I-011	C-LINK-CWORD	I-031	C-BNK-INSTAN
I-012	C-LINK-TRACE	I-032	C-BNK-VRBWRD
I-013	C-VRB-MXLINK	I-033	C-BNK-NOTFOU
I-014	C-VRB-2-LONG	I-034	C-BNK-E2LONG
I-015	C-VRB-2SHORT	I-035	C-BNK-TRAC-1
I-016	C-VRB-MAXLNK	I-036	C-BNK-TRAC-2
I-017	C-VRB-HEADER	I-037	C-BNK-SI-DUP
I-018	C-VRB-DLINCO	I-038	C-EVNT-LRIH?

[63] Ideally, of course, no events would be discarded at this stage; in practice, a rate of less than 1% is currently observed.

[64] Following the reformatter, the event treated by the filter will be in a flat ROOT format.

[65] These are data-driven: based on the current header, the reformatter knows how to check for the following header.

Errors are reported to log files, saved on the corresponding Processor node, as (# denotes time stamp)

```
/cdf/log/l3_node/l3_node.out_#
```

and are pre-scaled (the first ten are saved, and then 1 each 1000).

Errors are also sent to the external network, and are displayed in the *error logger* and in the *Level3 display* in the control room.

Reformatter-iii **How to decode reformatter errors**

Use reformatter decoder button in Ace Control Panel, by entering the information (namely a set of four numbers, identifying the partition, SCPU, VRB, Link) displayed both in the *Level 3 Display* for each Processor (click on a node number, and display corresponding error messages) or in the *Error Logger*. This way the offending sub-detector can be identified. [66]

Reformatter-iv **How to understand reformatter errors**

This is a difficult matter, and there will be no attempt of describing it here. [67]

Reformatter-v **What to do about reformatter errors once they start to happen**

In general, if the rate of reformatter errors is bigger than some threshold, currently agreed on 1% over the last $30s$, the respective expert (i.e. responsible for the offending system) should be paged.

The error is reported in two places: in the Level3 Display, where the accumulated reformatter rate for the run is indicated, that number will turn red; also, a warning 'Orange Window' will be popped-up by RC (again based on the instantaneous rate, whenever it passes the threshold) containing explicit instructions for the shift crew.

[66] In case a fib channel is identified, the corresponding ladder will have to be marked offline for the rest of the store by a silicon expert.

[67] If the problem (after reformatter decoding) points to silicon, it will most probably not be related to EVB. But there may be problems e.g. in VRB headers; or in VRB channels, suggesting in this case FE problems.

3.3 L3 filter

L3 filter-i **Who is the group of people responsible for it**

The University of Oxford group.

L3 filter-ii **How it comes into L3 farm**

The first step of distributing the filter packages is to move the corresponding three tarballs from an online computer [68] to b0l3gate2 (which occurs during *cold start* transition). From there, the distribution over the Processor nodes is made with the **l3_byteline** program.

L3 filter-iii **The different packages: filter, calib, tcl**

The filter is run on the reformatted event. It has common features to the offline analysis framework, uses AC++, tcl input, defining also *trigger paths*.

The package structure contains tcl, calib(ration), and filter packages.

L3 filter-iv **What are trigger tables**

A *trigger table* is a specification of a set of *trigger paths*. A trigger path is a unique combination of consecutive L1, L2, and L3 triggers. For a given event, a trigger bit is set for each trigger; the event will satisfy a trigger path if it satisfies the AND of the corresponding L1, L2, and L3 trigger bits.

Past Level3, different trigger paths will feed a unique dataset; datasets will be written to various *streams*.

L3 filter-v **Where are filters found on l3 nodes**

Each Processor node has its own copies of the filter packages (a few versions are kept), archived in the form of tarballs in the directory

```
/cdf/level3/filter/tar/
```

and unpacked in the directories

```
/cdf/level3/filter/bin/
/cdf/level3/filter/calib/
/cdf/level3/filter/lib/
/cdf/level3/filter/tcl/
```

[68] This computer is b0dap31, from which the $/cdf/code - level3$ directory in b0l3pcom2 is mounted from.

L3 filter-vi **How l3_node starts filter and how it communicates with it**

Each of the two analysis chains, which are part of l3_node in the Processor nodes, contains a filter interface module. It communicates, through message queues, to the L3 I/O interface of the filter executable, *L3exe+* (see Figure 13).

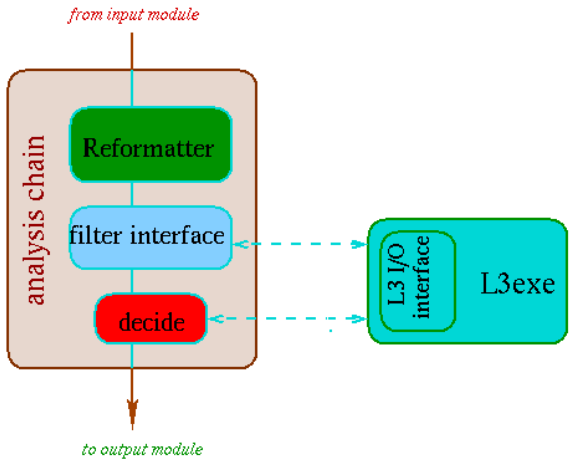


Figure 13: l3_node — filter communications.

L3 filters are started by the dataflow executable via executing a shell script

$\$L3\_CSHDIR/l3\_start\_filter$ +flags

The filters run as separate Linux processes, distinct from the dataflow process. A L3 filter requires that a standard CDF Offline environment is initialized, and all necessary setups are done in the l3_start_filter script, including all environment variables and pointers to shared libraries that came with the filter tarball.

L3 filter as a standard Offline AC++ executable has an input and output module, which in case of L3 are Level3Input and Level3Output modules. These input and output modules of the filters communicate with the analysis modules in dataflow executable l3_node. Each chain of l3_node contains one filter analysis module and each filter analysis module communicates with its filter executable process, passing event back and forth. The events themselves are kept in a large shared memory segment and are available to both dataflow and filter processes at all times. Message queues between l3_node and L3 filters allow l3_node to pass commands and get filter decisions back.

L3 filter-vii **Where to find info if filters fail to start**

Such information can be found either in l3_node or L3exe log files.

L3 filter-viii **Where are filter log files, how to tell if a filter has crashed or not**

Filter log files are located in each Processor, in /cdf/log/l3_node_exe_# .

L3 filter-ix **What happens if filter crashes, what about core files**

Filter crashes generate core files in the corresponding Processor nodes.

A crashed filter disables its analysis chain but does not kill l3_node on the affected Processor node. The dataflow executable detects filter crash and generates an error message sent to L3 display and error logger. At the same time the event that crashed the filter is marked b̈adb̈ut nevertheless is sent to CSL to be saved in a special file. If both filters have crashed on a Processor node, the l3_node stays alive but events do not go through this processor anymore. Problems such as crashed filters are always indicated on L3 display.

At the end of the run, l3proxy itself saves some of the possibly existing core files, which are automatically sent, together with the log files, to online computers via NFS; and an email is automatically sent to the L3 filter group.

3.4 Relay

Relay-i **What is relay**

The primary purpose of the Relay system is to execute commands on remote computers – e.g., by l3proxy over the internal nodes of the L3 farm.

Relay-ii **What is orbacus/CORBA**

Corba (acronym for *Common Object Request Broker Architecture*) is a general standard for working with distributed objects, i.e. an infrastructure that computer applications use to work together over networks. It allows the interconnection of objects and applications, regardless of the programming language of the applications, machine architecture, operating system, and network. An application is in servers that must handle large number of clients – as is the case of the Level 3 farm.

A client accesses an object by issuing a request to the object. The intermediary is referred to as an Object Request Broker (ORB). ORB negotiates between request messages from clients and object servers. Clients request services from objects (a.k.a. servers) through a well-defined interface, specified in IDL (Interface Definition Language). CORBA separates interface from implementation providing language-neutral types that make it possible to call objects across language and operating system boundaries. Using CORBA IDL, it is possible to make existing code look like an object on the ORB, even if it's implemented in, say, C++. Basically the code can be written in any language and then can be connected through ORB.

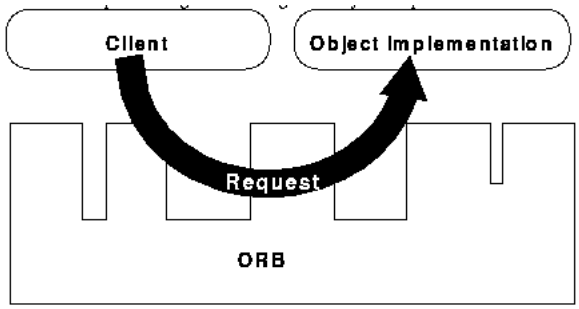


Figure 14: CORBA communication: client sends request to server through ORB.

There are various CORBA compliant ORB s which support various platforms and programming language mappings. E.g. ORBacus (*OmniBroker*), offers C++ and Java mappings (used in level 3 farm), ILU (*Inter-Language Unification*) includes IDL compilers for C, C++ mappings and supports VxWorks (used in EVB).

Orbacus exists in all L3 PCs, its UPS package being OOC; on the Gateways a code generator is run while elsewhere shared libraries are used. Client, server objects, built for our conditions, are included in CVS/UPS package *Relay*, located in $/cdf/level3/filter/relay$.

This also is the framework on which the Monitoring system is based.

Relay-iii **What is relay map**

This map defines the network architecture. It is generated automatically by l3proxy from Hardware database.

It can be found in l3proxy and relay log files; e.g.:

```
b0l3gate2> cd /cdf/log/l3proxy
b0l3gate2> less l3proxy.out_# | grep map
```

Relay-iv **Which processes have to be found on which nodes**

The two main relay processes are denoted *RelayService* and *RemoteServer*; the former is responsible for forwarding requests to the following node, while the latter executes commands locally.

RemoteServer exists in every single internal node of the farm (Processors, Converters, Outputs).

RelayService is run on Output node b0l3u01 (which provides forwarding to the other Output nodes and the Converters) and on the Converter nodes (to the Processor nodes in the sub-farm).

Gateway2 contains the relay client object, built in Level3 proxy.

To check for running relay processes one can use something like [69]

```
b0l3gate2> ssh b0l3u01  'ps auxwww  | grep Re'
or
b0l3gate2> $L3_CSHDIR/l3_all_nodes_exe +out stdout
          'hostname; ps auxwww | grep RemoteServer'
```

The log files – which are created each time a new relay is started – are located, on each Processor node, in

```
/cdf/log/l3_relay_server/RemoteServer.out_#
```

Relay-v **How long it takes to run a command on the entire farm**

It takes $\sim 15s$ to run a simple command on the farm and return.

[69] Can also check **netstat -p**.

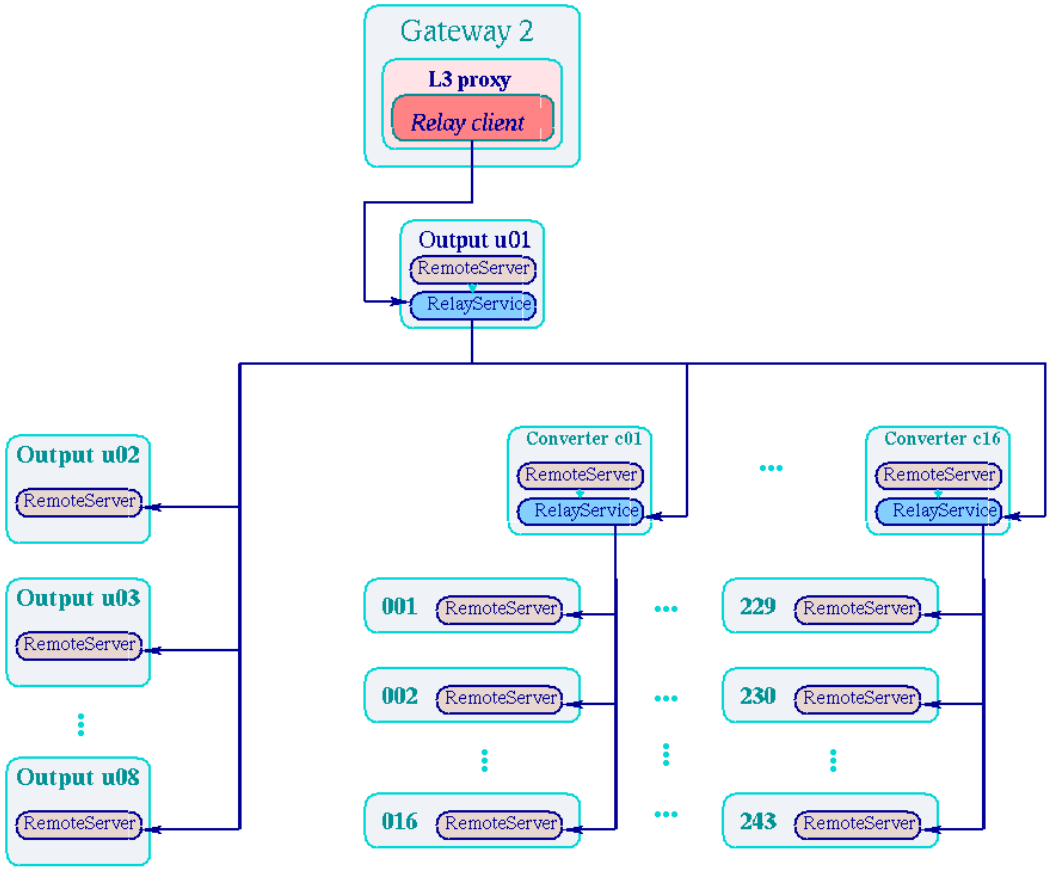


Figure 15: L3 Relay system.

Relay-vi **Where relay is used**

Relay is currently used for two purposes. It is used by l3proxy to run the farm during data taking, and to distribute code (executables, etc) over the farm.

Relay-vii **How to restart relay from clean state and when it is needed to be done**

Relay is started from a clean state by starting l3proxy after exiting relay processes

```
$L3_CSHDIR/l3_all_nodes_exe +out stdout  'killall RelayService RemoteServer'
```

This will be necessary whenever (i) environment variables of the l3proxy account have been modified, (ii) hardware database has been changed, or (iii) l3proxy gets stuck for unknown reason on starting up.

Relay-viii **If relay does not start, how to find out why; trace the problem going along the relay tree, finding failed mode, how to restore it**

Failures may occur whenever e.g. a node hangs or an executable crashes. A suggested procedure should be to descend through the relay map, explicitly check if *RelayService* and then *RemoteServer* exist in the corresponding nodes. Once identified the first problematic node along the relay map, one should try to understand the problem itself – e.g.check network connections with netstat command, ... (otherwise, rebooting the Processor node may be a last rescue alternative).

It will be necessary, or convenient, sometimes to kill the relay processes, e.g.

```
b0l3gate2> $L3_CSHDIR/l3_all_nodes_exe +out stdout
          'killall -9 RelayService RemoteServer'
```

which will then need to be re-initiated, done simply by re-starting l3proxy (from Ace Control Panel).

If a node is down for whatever reason and cannot be restored, it should be marked offline in the hardware database, so that Relay will not get stuck on it.

3.5 Level3 proxy

L3 Proxy -i **What it is for**

Level3 proxy provides the connection between the Level3 internal farm and the outside world. It is a process constantly running on b0l3pcom2.

Effectively it runs the Level3 farm. It implements the *states machine*, as instructed by *run control* (RC), issuing appropriate commands over the farm using the Relay system. It is based on ROOT. Uses ORBACUS for internal and SmartSockets for external communications.

L3 Proxy -ii **How to check if it is alive, where are the log files, understand the log files**

It can be checked that it's alive from the Ace Control Panel. Alternatively, one can also check L3 Display and L3 'health bar' on the main DAQMON window.

The log files are located in Gateway2, in

```
/cdf/log/l3proxy/l3proxy.out_#
```

L3 Proxy -iii **What is the conceptual structure of the program**

Its structure is formed of *communications*, *actions* and *relay* parts (i.e. objects); see Figure 16.

Figure 16: Level3 proxy structure.

L3 Proxy -iv **How l3proxy is related to ROOT**

Its classes are derived from ROOT's *TObject* class. SmartSockets usage is hidden (encapsulated) from ROOT to avoid linking conflicts.

L3 Proxy -v **Communication between Run Control and l3proxy (transition and configuration messages)**

Run Control sends two messages in sequence for Partition and Cold/Warm Start. Namely: *SetPartition* and *L3ReadoutList* [70] for Partition; transition message and *L3ReadoutList* for Cold/Warm Start; transition message only on any other transition.

L3 Proxy -vi **Heartbeat and ping error messages**

Messages, denoted *heartbeat*, are periodically sent from l3proxy to RTServer. The objective is to confirm that l3proxy is alive. These messages also contain information about the partitions, and are responsible for the partition status bars (with colors green/yellow/purple, and last heartbeat) at the bottom of the L3 Display.

All the nodes of the L3 farm are also ping-ed regularly; if a node cannot be ping-ed, it will appear in black on the L3 Display.

L3 Proxy -vii **What is done by l3proxy during the state transitions**

The start up actions performed by l3 proxy are:

1. Connect to the hardware database
   Read all nodes that are online, i.e. the L3 farm configuration
2. Connect to RTServer, and listen to transition messages from RC
3. Start relay throughout the farm

The l3 proxy actions performed during each state transition are as follows.

a) Partition transition

1. clean-up nodes to be used in the partition
2. start monitoring: *l3_mon_reporter* (on all nodes)
3. start dataflow executable, *l3_node* on Converter and Output nodes
4. start additional monitoring programs (*EventFunnel*s, *l3_mon_client*s)
5. connect all monitoring into one net
6. reply to RC

These actions are coded into the *actions* part of l3 proxy.

b) Cold Start

1. handle *tcl* package
   - download package from online computers to the Gateway [71]
   - distribute it over all Processor nodes

[70] Which contains information relative e.g. to run, reformatter, and filter parameters.
[71] $b0dap31 : /cdf/code - level3$, NFS-mounted.

- expand[72] tarball

2. handle *calib* package
   As above.
3. handle *filter* package
   As above.
4. start l3_node on Processor nodes, and final monitoring pieces (that connect to these nodes) [73]

c) Warm Start

1. check existence of tcl and filter packages, and distribute *calib* package
2. start l3_node on Processor nodes, as well as missing monitoring

d) Activate Do nothing; just change state variable.

Note: This is similar to *Ready* and *Idle* states.

e) End

1. end l3_node on Processor nodes (via msgQ's)
2. collect *end of run* (EoR) summary information, and send it to RC
3. collect possibly existing L3 filter core files, which are to be automatically sent to L3 software experts

f) Abort

1. end l3_node on Processor nodes
2. collect run summaries [74]

g) Reset

1. stop l3_node on Converter and Output nodes
2. stop all monitoring processes (on Converter, Processor and appropriate Output nodes)
3. final cleanup (close everything in all nodes)

L3 Proxy -viii **What is End Of Run summary**

During End or Abort transition, l3proxy gets statistics from all the nodes, puts them together and sends such information to Run Control, which will then save it to database.

The run summary is also saved in the l3proxy log file; below is an example. [75]

[72] And check.

[73] This is the step that most typically may give problems.

[74] Core files are not collected, as occurs in *End* transition.

[75] Statistics for Output nodes are currently disabled; the fact that an Output node can belong to two different partitions makes the process troublesome in this case.

```
----------------------------------------
Run Summary
----------------------------------------
  run number                     : 142178
  partition                      : 0
  converter nodes                : 15
    number of chains per node    : 2
    number of analyses per chain : 1
    input                        : 5119
    input errors                 : 0
    output                       : 5119
    output errors                : 0
  processor nodes                : 215
    number of chains per node    : 2
    number of analyses per chain : 2
    input                        : 5119
    input errors                 : 0
    reformatter pass             : 5119
                fail             : 0
    filter pass                  : 5119
           fail                  : 0
    output                       : 5593
    output errors                : 0
  output nodes                   : -1
    number of chains per node    : 0
    number of analyses per chain : 0
    input                        : 0
    input errors                 : 0
    output                       : 0
    output errors                : 0
----------------------------------------
```

L3 Proxy -ix **How to trace a problem along the chain Transition Failure, l3proxy log file, Relay log files, local CV/PR/OUT log files and process status**

This is more of a practical skill that is difficult to describe in full.

From the l3proxy log file it should be apparent what component of L3 farm is failing. One has to descent via log files to the component that is giving the trouble.

For example, if one fails to get a *filter/calib/tcl* package from the online machines, the problem is already visible from messages in l3proxy log files. If a problem occurs on a particular node, one has to go deeper into the farm, and analyze it locally. l3proxy may report a relay failure associated with a particular node; this kind of problem is easy to understand as well, most often the responsible node simply has a crash of OS.

Slightly more confusing is failure to start on Partition or ColdStart while the responsible node remains alive and well. In that case, one has to go to the RemoteServer

log file on the responsible node and check for the output of the command

$$\$L3\_CSHDIR/l3\_node\_check\_running\ REP\ CON\ MON$$

This command, initiated by l3proxy during a state transition, verifies that this node - whether it is Converter, Processor or Output - has started properly all its corresponding executables. The three arguments correspond to checking:

- **REP**: *l3_mon_reporter*, a part of monitoring system that sends out info/error messages from l3_node
- **CON**: *l3_node*, the dataflow executable
- **MON**: *l3_mon_server*, a part of monitoring system sending info on state of dataflow every 4 seconds

If the result of any of the above checks is non-zero, it indicates problems. Further understanding of the problem can be gained from the analysis of the appropriate log files: */log/l3*$_n$*ode/* directory for *l3_node* and */log/l3_box_monitor* directory for *l3_mon_reporter* and *l3_mon_server* programs.

3.6 L3 monitoring

L3 monitoring-i **The general scheme**

The L3 Monitoring system is based on CORBA. On the Gateway2 monitoring processes are permanently running, while on the internal nodes they are started for each run. The general structure of the Level3 monitoring system is depicted in the figure below.

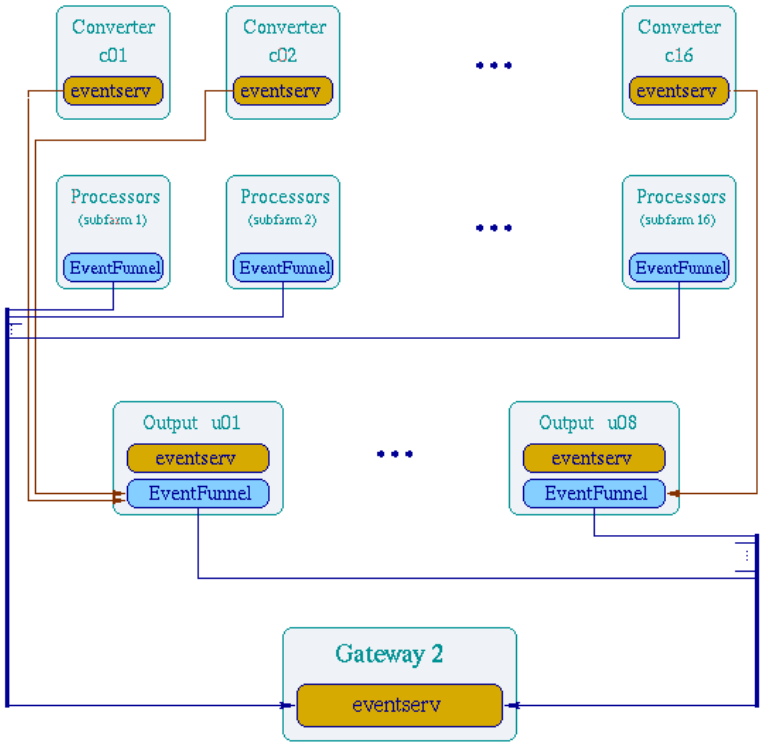


Figure 17: Level3 farm Monitoring.

L3 monitoring-ii **What programs run where**

The processes **l3_mon_reporter**, which itself spawns **eventserv**, and **l3_mon_server** are run on all internal (Converter, Processor, Output) nodes. The *eventserv* executable is based on a standard CORBA implementation, that allows to receive and send messages from/to many input/output sources via CORBA; it possesses queues which are characterized by a FIFO structure.

On the first Processor node of each subfarm, there exists additionally **EventFunnel**. Its purpose is to slow down the rate of messages. While its input is asynchronous, it has a periodic output, sending all 4*s* its received, accumulated messages. The *eventserv* and *EventFunnel* processes are connected together by **l3_mon_connect**;

this latter process is started by l3proxy in every node when appropriate and is later ended, it is not found permanently running.

On Gateway2 there is **l3_mon_reporter**, with its **eventserv**, and **l3_mon_client**.

l3_mon_client runs on Gateway2 and actually on all other nodes as well. It receives from *eventserv*, does some sorting of the messages and then prints them to *stdout*. On the internal nodes this *stdout* is directed to a log file. On Gateway2, it is piped into the *stdin* of *MConverter*.

MConverter is a java program, permanently running in Gateway2, that takes messages from the standard input and creates SmartSockect messages in *Merlin* format.

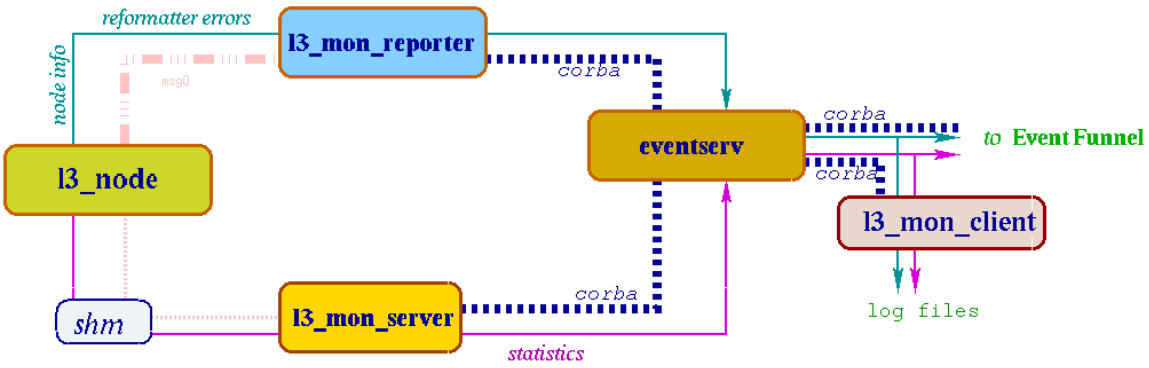


Figure 18: L3 Monitoring processes on an internal node.

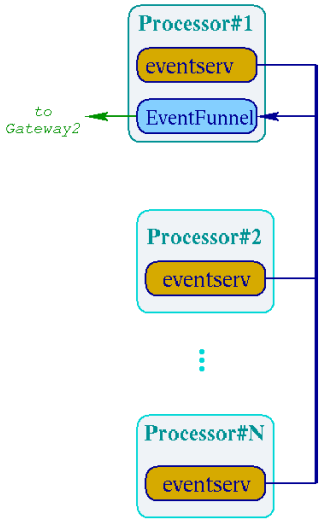


Figure 19: L3 Monitoring processes on a subfarm.

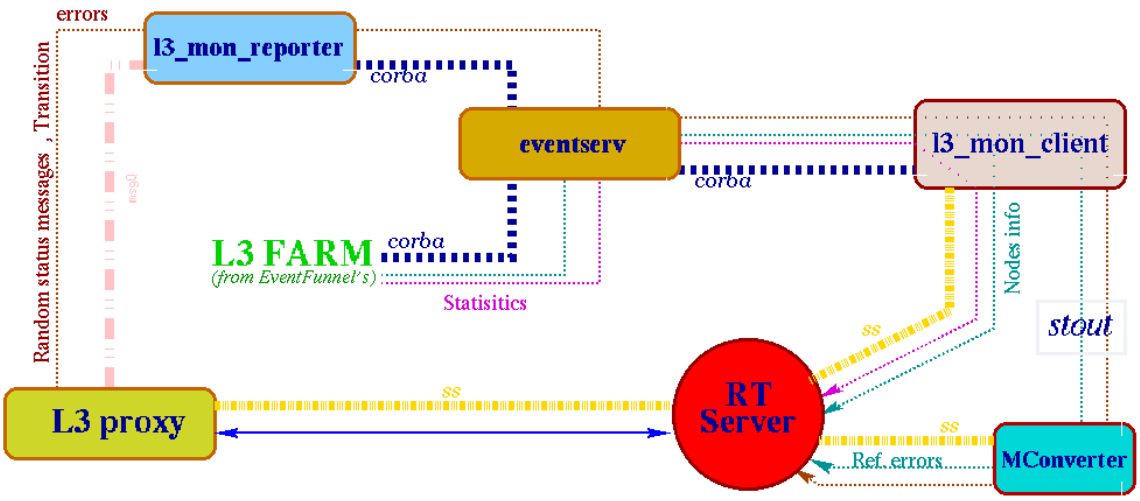


Figure 20: L3 Monitoring processes on Gateway2.

L3 monitoring-iii **What is the starting order, how connections occur**

In general, servers are firstly started, to which clients connect. The sequence follows:

1. *l3_mon_reporter*, with *eventserv* (on all nodes)
2. *EventFunnel* (on appropriate Processor nodes)
3. start l3_node dataflow executable (on Converter and Output nodes)
4. *l3_mon_server* (on Converter and Output nodes)
5. *l3_mon_connect* (on all nodes, at beginning)
6. l3_node dataflow executable (on Processor nodes)
7. *l3_mon_server* (on Processor nodes)

The sequence of actions identified in the items above occurs during the different transitions as follows:

- *Partition*: start 1, 2, 3, 4, 5
- *Warm start*: start 6, 7
- *End*: stop 7, 6
- *Abort*: stop 7, attempt to stop 6 (otherwise kill it)
- *Reset*: stop [76] 5, 4, 3, 2, 1

[76] Includes additionally the actions that take place during *Abort* state.

L3 monitoring-iv **Where are the log files, what can be learned from them**

Log files are created by all monitoring executables, and are found in the directory

```
/cdf/log/l3_box_monitor
```

In particular, the following log files exist on the appropriate nodes: [77]

```
/cdf/log/l3_box_monitor/l3_mon_reporter.out_#
/cdf/log/l3_box_monitor/l3_mon_server.out_#
/cdf/log/l3_box_monitor/l3_mon_client.out_#
/cdf/log/l3_box_monitor/EventFunnel.out_#
```

L3 monitoring-v **How to check if monitoring for a given node is working properly**

To check the monitoring for a node one can look first to the L3 Display, and then login to the node and check is all expected processes are up. The next step would be start looking through the log files.

[77] l3_mon_reporter.out_# contains log information of both *l3_mon_reporter* and *eventserv*.

3.7 L3 proxy account

Proxy account-i **Log in**

The directory of *l3proxy* account is, together with the entire home directory, nfs-mounted from b0l3pcom1 in the entire farm.

In order to log in, as usual, one needs to go first to one of the Gateways, from any online machine; from there the l3proxy user can automatically login, with no password required, into any other node of the L3 farm. The following is an example.

```
b0dau30>  ssh -l l3proxy b0l3pcom2
Logged in to l3proxy account
Setting environment variables
...
b0l3gate2>  ssh b0l3u01
Logged in to l3proxy account
Setting environment variables
...
b0l3u01>
```

Proxy account-ii **What environment has to be set, .env file, dependence of environment on hostname**

The environment for the account is set by the *.env* file. Here Gateway1, Gateway2 and all the internal nodes are set separately (and differently).

The primary difference here between the Gateways and the internal nodes is the location of the level3 and relay code, and required UPS products, that are stored on local drivers on each node. Another important distinction arises from different Linux kernel versions used in Gateway1/Converters and Gateway2/Outputs/Processors, which requires different environments and setups.

Proxy account-iii **Where is level3, reformatter and relay code**

On the Gateways, the level3 and relay binaries are located in

$\sim l3proxy/checkout/level3$

$\sim l3proxy/checkout/relay$

while reformatter is properly linked in the former.

On the internal nodes (as seen before) the code is located in

$/cdf/level3/filter/$

Proxy account-iv **What is local distribution of the code and how to do it**

Specified level3 and relay code versions (denoted below as vx_y and vw_z, respectively) can be distributed (after being successfully produced, with $L3_GMKDIR/**gmake**) over the farm, using the *l3_byteline* program, [78] from Gateway2, as follows.

```
b0l3gate2> cd $L3_GMKDIR
b0l3gate2>  gmake distribute [DL3VER=vx_y] [DRLVER=vw_z]
```

The following sequence of actions take place in the process:

1. create tarballs
2. start relay [79]
3. distribute with *l3_byteline*
4. expand tarballs (in all nodes)
5. stop all relay executables on the farm

In order to guarantee that the *old* relay processes were exited, one does as follows

```
$L3_CSHDIR/l3_all_nodes_exe +out stdout 'killall RelayService RemoteServer'
```

The next step is to specify the new version to be used in .env file, and start l3proxy.

A specified version of the code can as well be removed using

```
b0l3gate2>  gmake remove [DL3VER=vx_y] [DRLVER=vw_z]
```

Note that for both *distribute* and *remove* either or both options *DL3VER* and *DRLVER* can be given.

Proxy account-v **What can be found in /cdf directory on all nodes**

On the Gateways, the /cdf directory contains various needed NFS-mounted directories from online computers. From b0dap30 various products are made available. Of particular importance here is the directory

```
b0dap31:/cdf/code-level3
```

containing the three filter packages.

On the internal nodes (where the only NFS-mounted partition is /home), the /cdf directory is a separate partition (/dev/hda1 on Processors), containing in particular the filter code (*/cdf/level3/filter/*), the log files (*/cdf/log/*), and UPS products (*/cdf/products/*).

[78] As is the case for filter, seen before.

[79] The current, soon-to-be-old version of Relay is of course used at this stage.

```
b0l3gate2% ls /cdf
 code-IRIX-#   code-Linux-#   code-common
 code-level3    evbproxy        level3
 log            lost+found      products
b0l3gate2% ls | grep cdf
 /dev/hdc10                     /cdf
 b0dau30:/cdf/code-common       /cdf/code-common
 b0dau30:/cdf/code-IRIX-#       /cdf/code-IRIX-#
 b0dap30:/cdf/code-Linux-#      /cdf/code-Linux-#
 b0dap31:/cdf/code-level3       /cdf/code-level3
b0l3gate2% ssh b0l3100 ' ls /cdf'
 level3   log   products   sdr   upgradeutil
```

Proxy account-vi **What is base key, what kinds are defined**

In the original design of the level3 system it has been foreseen that different users can run *level*3/*relay*/*monitoring* programs at the same time without conflicting with each other. Each program of *level3 package* uses a predefined offset for IDs when creating/connecting-to message queues and shared memory segments, as well as in determining the network port numbers to connect/listen to. There is one environment variable defining this constant offset for level3 dataflow and monitoring programs, $L3\_BASE\_KEY$, and another one for Relay connections, $RELAY\_BASE\_KEY$. Both of these variables are set in ~/.*env* file of the l3proxy account. For the l3proxy user both offsets are normally set to $0x4000$.

EVB & LEVEL3 COMMON

4 A few topics common to both EVB/L3

4.1 Raw data format

Raw data format-i **Where data is seen in raw format**

On SCPUs, Converter, and input to Processor nodes.

Raw data format-ii **How it is different from offline (roughly)**

The structure of raw data is dictated by the detector readout; it is composed of organized super-sets of data fragments reflecting the sequence *part of sub-detector, VRB, SCPU, whole event.*

On the other hand, offline data is composed of banks by sub-detectors (e.g., all-COT bank, all-SVX bank, ...).

The main change in the format of data occurs at the level of the reformatter on the Processor nodes. The sets of SCPU fragments constituting a single event provided by the Converter node are reformatted into TRYBOS format, and delivered in ROOT format after passing the filter.

Raw data format-iii **The exact details of VRB fragment structure for SVX and non-SVX VRB crates**

The VRB fragments are composed of an header and links; the links themselves are made up of several banks, an header and a control part at the end. [80]

Difference between SVX and DAQ formats

Raw data format-iv **Event Builder checks of VRB structure and most common errors**

The EVB performs various data quality checks, namely upon the VRB fragments, among which the following.

- Consistency of event ID and control sequence at the end of each link of the VRB fragment.

  Merlin/EVB mnemonics: SCPU_TRACER_EVENT_ID, SCPU-P0-E-TracerEventId.

- Compatibility of VRB fragment total length with the sum of the lengths of its channel components.

  Merlin/EVB mnemonics: SCPU_BAD_CHANNEL_COUNTS, SCPU-P0-E-BadChannelCounts.

- Whether the total length of the VRB fragment is greater than $32B$, less than $65528B$, and divisible by 8.

[80] See example in the Appendix.

Merlin/EVB mnemonics: SCPU_BAD_VRB_BYTE_COUNT, SCPU-P0-E-BadChannelCounts.

- Check on event size, $330KB$ being the buffer limit for a VRB.
  Merlin/EVB mnemonics: SCPU_BUFFER_OVERFLOW, SCPU-P0-E-BufferOverflow.

Raw data format-v **Where errors are reported, effect on data taking and what can be done about them**

Errors are reported via *Zephyr* messaging. On the Gateway, ZMConverter also sends some of them to RTServer (via SmartSockets). The messages appear in the Zephyr display and Error Handler.

Advanced

Raw data format-vi **How to do octal dumps of raw data files**

Raw data files, e.g. saved by the reformatter, can be dumped using **od**. For a specific example, od -v -t x4 data.raw | more.

Raw data format-vii **The format of the full raw data file**

In general, a **full event** is composed of the event header (includes message header and event length) and of several *SCPU fragments*.

A **SCPU fragment** is composed of the SCPU header (includes SCPU length and other SCPU info) and of several *VRB fragments*.

A **VRB fragment** is composed of the VRB header (includes VRB byte count and status, link lengths) and of several *link fragments*.

A **link fragment** is composed of link header (includes link length, single word), of several *banks* and of *link control words*.

A detailed explanation of DAQ VRB output data format is presented in the Appendix.

Raw data format-viii **Be able to find errors in full files with corrupted data saved by reformatter**

Use octal dump ('*od*') on the raw data files, together with the raw data format example given in the Appendix.

4.2 EVB/L3 Ace control panel

Ace control panel-i **How to start**

Having logged in as evbproxy user in b0l3pcom1,

```
[evbproxy@b0l3gate1 ~]$  ace
[evbproxy@b0l3gate1 ~]$ which ace
ace:     aliased to source /cdf/evbproxy/ace.csh
```

With ace privileges only, the procedure is to simply type **EvbControl**, from b0dap57, b0dap58 or b0dap59 as user cdfdaq.

```
b0dap57:~>   EvbControl
b0dap57:~> which EvbControl
EvbControl:      aliased to ~/bin/EvbControl.exp
```

Aces are also allowed, from b0dap57 as user cdfdaq, to login to b0l3pcom1 in a specified way,

```
b0dap57:~>  setup kerberos
b0dap57:~> /usr/krb5/bin/ kinit -k -t /var/adm/krb5/cdfdaq.keytab cdfdaq/cdf/‘hostna
b0dap57:~>  xhost + b0l3pcom1
b0dap57:~>  rlogin b0l3pcom1 -l evbproxy
[evbproxy@b0l3gate1 ~]  setenv DISPLAY b0dap57:0.0
[evbproxy@b0l3gate1 ~]  ace &
```

Ace control panel-ii **Safe and unsafe operations**

All *checks* can be performed without consequences. Exiting the gui is also safe. Cleanup L3 mon can also be performed during the Active state; it affects only the monitoring tasks.

However, starting and stopping proxies (including cleanup L3 or EVB) involve stopping the run, and all partitions using EVB and L3 should be in Start state.

The actions implied by clean up EVB include stopping EVB proxy, resetting all SCPUs and SM, and restarting EVB proxy.

The actions implied by clean up L3 include stopping L3 proxy, killing monitoring and communication processes, and restarting L3 proxy.

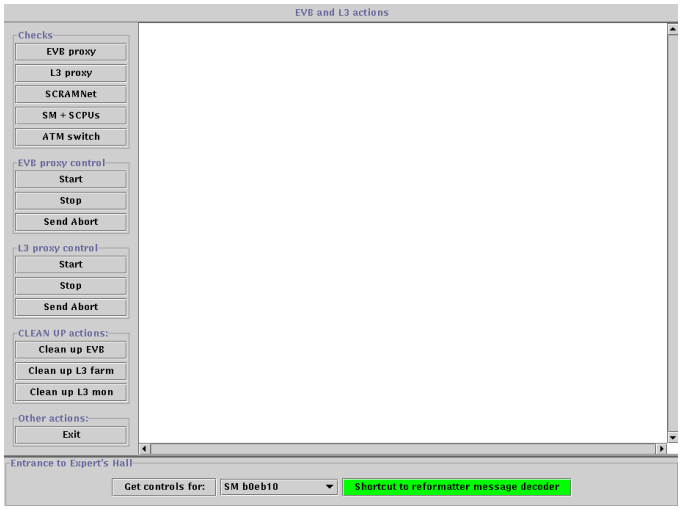


Figure 21: The EVB/L3 Ace Control Panel.

Ace control panel-iii **Possible failures and how to understand them (ace control panel does not start, buttons do not have any effect, etc)**

One thing to note for example is that an action can be performed only after the previous one has been completed. If the current action cannot be completed successfully, it will eventually time-out ($\sim$ 10 min.). In order to proceed otherwise, it will be necessary to kill a specific process (*ps auxwww | grep ops*) running on *b0l3pcom2*, exit and restart the application (as above). [81]

Ace control panel-iv **How to check the response to stop/start/cleanup L3 command with log files**

The relevant log files are located at

b0l3gate2 : /cdf/log/l3_box_monitor/l3_proxy_ops.out_#

The l3proxy log files (*b0l3gate2 : /cdf/log/l3proxy/l3proxy.out_#*) can be useful here as well.

Ace control panel-v **Where to find "reasons to clean up evb" in log files**

Any indications pointing to communication problems between processes existing on SCPUs and Converters (*tsend, builder, Boss*).

[81] Notice that buttons on the gui call a script, the script from b0l3pcom1 calls inetd of b0l3pcom2, inetd of b0l3pcom2 loads *l3proxy/.cshrc* and executes *...ops*.

Ace control panel-vi **The check for suspended processes and where the results are saved**

The check for EVB suspended processes can be performed via the *checks* button for *SM+SCPUs*. If any such process is found, a file is saved on b0l3pcom1 [82]

$$b0l3gate1 : /log/cdfevb/SuspendDump\_\#.log$$

Ace control panel-vii **Expert tools (e.g. configuration, statistics, hdwdb, atm test)**

These can be accessed through the buttons at the bottom part of the gui. It includes a *Shortcut to reformatter message decoder* and *Controls* for SCPUs and SM. The former allows one to use information given in reformatter errors (displayed e.g. on the Error Logger, or on the Level3 Display for the respective Processor node) – namely the four integers *partition identifier, SCPU, VRB, Link* – to identify the corresponding FE crate.

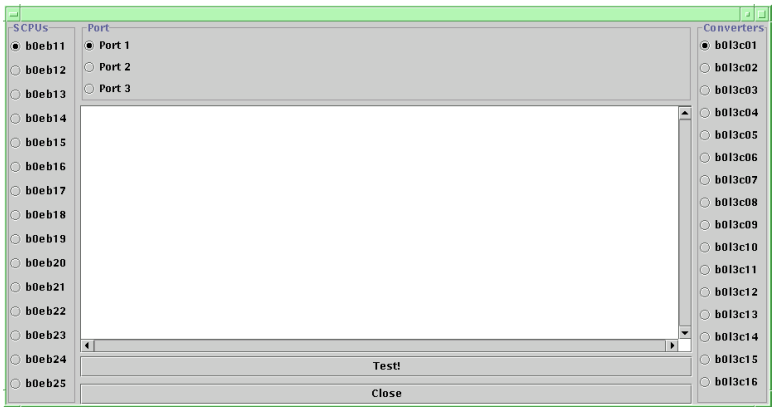


Figure 22: Expert's tool for testing ATM connections.

Also included is a module which has also been implemented for automatically test ATM connections between SCPUs and converters, Fig. 22. This is *not* a safe check: some messages are sent from SCPUs to converters and these messages will be interpreted as corrupted data.

Another tool here included is an utility for checking and changing the online database status of the Level3 nodes. It displays a map of the Level3 farm, Fig. 23, with the indication of the individual online flag of each node; it provides a straightforward way for changing their online/offline status, updating the database accordingly. To access this feature, the user must have an account on cdfonprd database with manager privileges.

[82] Additionally, an automatic email is sent to current senior EVB experts, Steve, Ilya.

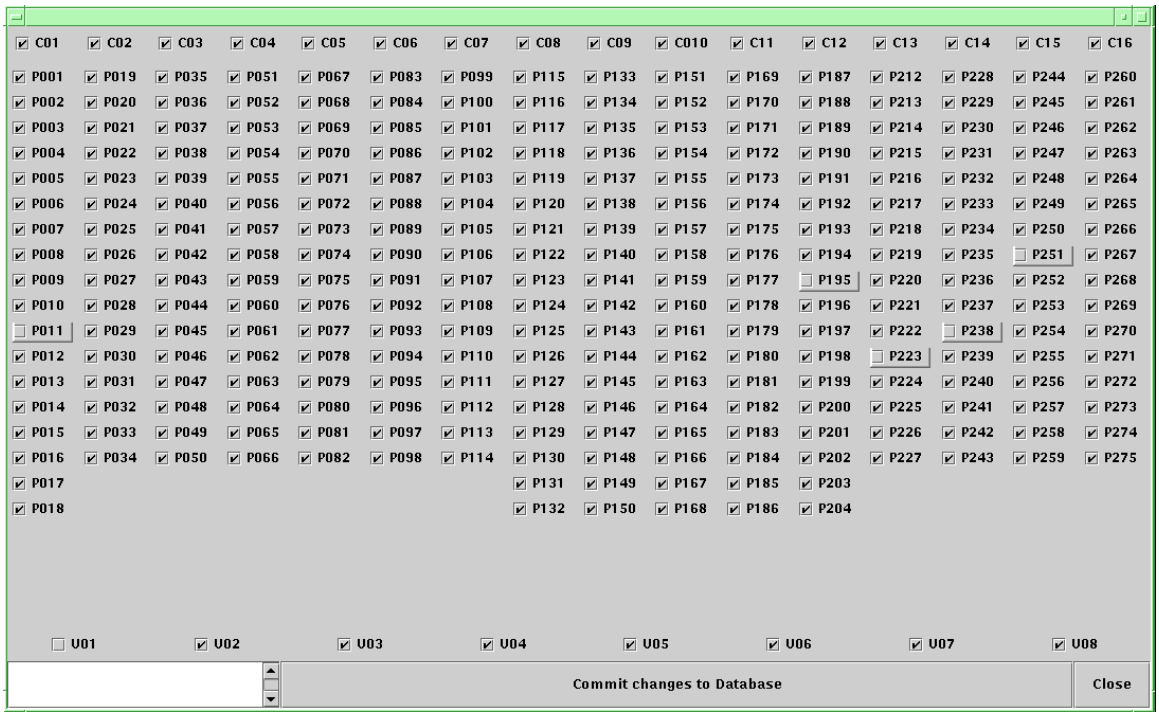


Figure 23: Expert's tool for checking and modifying the online status of nodes in the Level3 farm.

4.3 Monitoring GUIs

Monitoring GUIs-i **L3 display**

The L3 display is the main Level3 monitoring tool. It receives messages from Level3 monitoring processes [83] and translates that information in a visually representative, color coded fashion.

It is organized in three panels. The *farm monitor* describes the instantaneous (updated every 4 seconds) activity status of the various nodes (and respective analysis chains) of the farm. The *state/transition panel* shows the status of Level3 activity for each of the eight partitions. The *summary panel* displays various Level3 dataflow statistics.

In order to start the display, one needs to first start the daqmon gui if not running already,

```
b0dap#:~>  setup fer
b0dap#:~>  daqmon &
```

and from there click on the *L3* button.

[83] See section on *L3 monitoring* above.

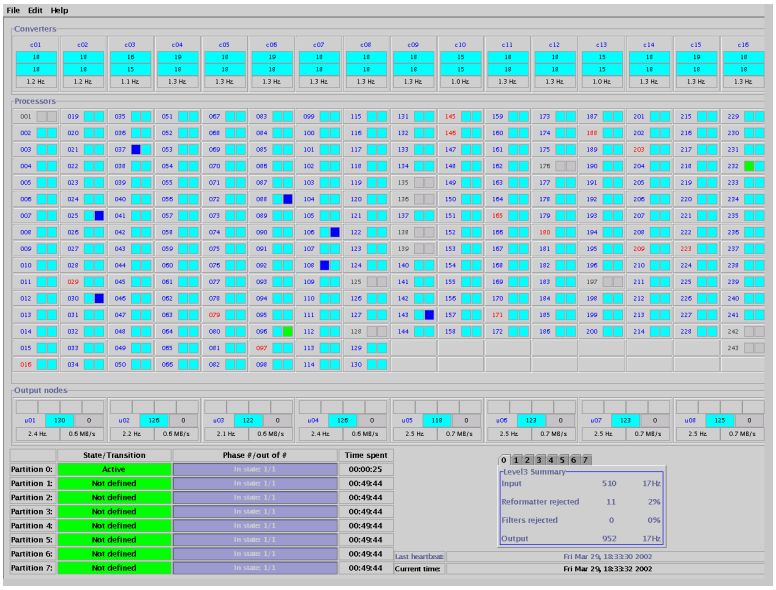


Figure 24: The Level3 Display.

Monitoring GUIs-ii **EvbDaqmon**

The EvbDaqmon is a EVB monitoring tool. It shows various information based on messages received from EVB monitoring processes.

The display is started from the daqmon gui.

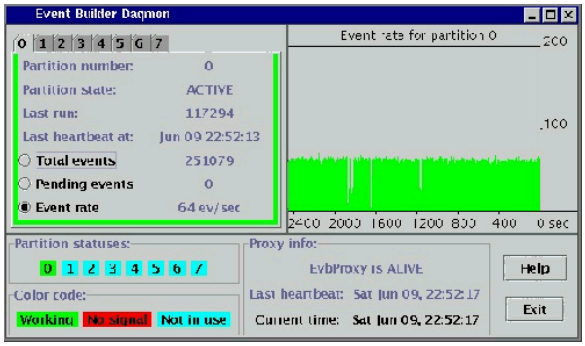


Figure 25: The EVB monitor.

DATA ACQUISITION

5 General DAQ topics

5.1 RC State Machine and Run Control

Run Control-i **What is state machine**

The process of data acquisition is organized in a set of sequential steps before the stage of actual data taking. These various stages are referred to as *states* (e.g. Start, Idle, Ready, Active), while the passage from one these states to the other are known as *state transition* (e.g. Partition, ColdStart, WarmStart, Activate).

The states and transitions taken together are referred to as *state machine*.There are several states machine defined at CDF. For data taking (physics) one uses *'DAQ state machine'*; also numerous calibration states machine are defined.

Run Control-ii **What happens with the detector/DAQ during transitions and in states of DAQ state machine**

While a state can be characterized as a steady stage of the DAQ system, a state transition corresponds to a transient state. The global sequential requests (by/to the different detector/DAQ sub-systems involved) that occur during state transition are transferred and coordinated by Run Control (RC).

Run Control-iii **What is a partition**

Partitioning is a concept that allows for different sets of parts of the system to work in parallel, independently of the others. This allows for the possibility of testing/calibrating several subsystems at the same time.

There are eight hardware partitions (numbered 0 to 7). [84]

Different L3 subfarms can be used in different partitions (a pair of subfarms formed by association to a common Output node cannot however belong to distinct partitions). Also different VRB crates may be booked by different partitions.

Run Control-iv **What is resource manager, booking resources**

The resource manager is responsible for the booking of resources (individual front end crates, L3 and EVB sub-systems) for each partition.

Run Control-v **How to configure a simple run** [85]

1. Setup the fer (front end readout) package containing RC code — setup fer
2. Start RC gui — rc &

[84] There also *software partitions* that do not talk to the Trigger Supervisor.

[85] See
http : *//www* – *cdfonline.fnal.gov/ace2help/runControl/rc_ace_guide/run_control_ace_guide.html*

3. Select DAQ from RC Enable menu
4. Select Partition from RC Partitioning menu, and choose one of the available partition identifiers
5. Select Run Configuration from RC Parameters menu, to choose a pre-defined Run Type, e.g. COSMICS; if one chooses to configure a run, select $Editor View Run Settings$ from RC Parameters menu, to view/edit the run parameters
   - select $TM enabled$
   - select $VRB(HardEvb)$ as Output Type
   - select the desired L3 subfarms
   - set convenient values of additional parameters relevant to L3 configuration in the table below that
   - select FE crates from CrateSets

The mentioned (additional) L3 parameters that can be set in RC are used to specify paths for data flow in the L3 farm (recall discussion on Level3 dataflow mechanics). In particular, one may specify which output and input $l3\_node$ modules are to be used by the various internal nodes.

Run Control-vi **What is Error Handler**

The Error Handler (EH) cooperates with RC allowing for the identification of possible causes of a failure. It displays information messages (Merlin format) from the systems participating in the partition. [86]

Normally, a EH is started automatically by RC when a new partition is booked. Otherwise, the monitor can also be explicitly started from RC by selecting Start Error Handler from RC Partitioning menu; or

```
> setup errmon
> startEH #
```

where # is the partition identifier.

Run Control-vii **The use of "Reply and Acknowledgements" window**

The Reply and Acknowledgements window shows the status of the various systems booked for the current partition, namely its (color-coded) status relative to a current state transition.

[86] To watch a different partition need to start another Error Handler display.

5.2 CDF trigger system

Trigger system-i **How it works in general**

The CDF trigger is a three-level, pipelined, buffered system. While Level1 combines some calorimeter, muon and COT information, a Level2 decision involves more precise and further information of the detector subsystems. A Level2 acceptance of an event initiates its full detector readout. When such Level2 acceptance signal is communicated (by the Trigger Supervisor, TS) to each Front End crate, data (mini-fragments) are loaded from the Readout Boards to the Tracer buffers. If this occurs successfully a '**done**' signal is sent to the TS, and data is directed to the VRB board. It may happen that the VRB buffer has not enough free space in which case a '**busy**' signal is generated. Otherwise, when 'done' signals from all FE crates are reported to the TS, this instructs, through the *Trigger Manager* (a TS process), each SCPU to obtain the event data (VRB fragments) from its associated VRBs and send the resulting SCPU fragment to Level3. A Converter node on the Level3 farm receives such SCPU fragments from all SCPUs, at which point it contains the full event data, and forwards it to an available Processor node. There the event is reformatted, and processed by the Filter. A successfull event is conducted to the associated Output node, and transmitted to CSL.

A triggered event has passed a set triggers at each level and was correspondingly associated with a set of trigger bits. A set of Level1, Level2, Level3 triggers defines a Trigger Path (which is potentially associated with a certain type of Physics). Different Trigger Paths compete in parallel, and a set of which defines a Trigger Table (this is what is specified by RC while configuring the run).

Trigger system-ii **Done dead time, done timeout**

Done dead time and done timeout are associated with a FE crate readout failure. If data cannot be loaded from the Readout Boards to the Tracer buffers in a FE crate, this generates **done dead time.** If it fails to succeed within a specified (RC) time window a '**done timeout**' is reported to the TS which then stops the run.

Trigger system-iii **Busy dead time, busy timeout**

Busy dead time and busy timeout are associated with EVB not processing the incoming events fast enough as it would be required to keep its buffers with available space, otherwise preventing data from flowing from the FE (Tracer) buffers to the VRBs. Such 'busy' conditions lead to FE dropping events and contribute to **busy dead time**. Whenever such 'busy' signals last for more than a specified (by RC) time window a '**busy timeout**' signal is reported to the TS which stops the run.

These can be caused by high L2 accept rates, internal EVB problems, or output difficulties of the systems that follow (down the data stream).

Trigger system-iv **TSI dead time**

TSI dead time corresponds to TS internal delays.

5.3 Web support

Web support-i **Electronic log books**

The address of the CDF Electronics Logbook's page is

$$http://www-cdfonline.fnal.gov/e-log/$$

It contains links to the main *CDF Shift E-Log*, as well as to the *L3* and the *EVB* e-logs.

Web support-ii **Help pages**

The EVB/L3 page for experts, also including the contents of this document, is available at

$$http://www-cdfonline.fnal.gov/evbl3shift/evbl3pager.html$$

The EVB/L3 help page for ACEs is located at

$$http://www-cdfonline.fnal.gov/evbl3shift/evbl3shift.html$$

(the DAQ ace information page contains a link pointing to this address as well).

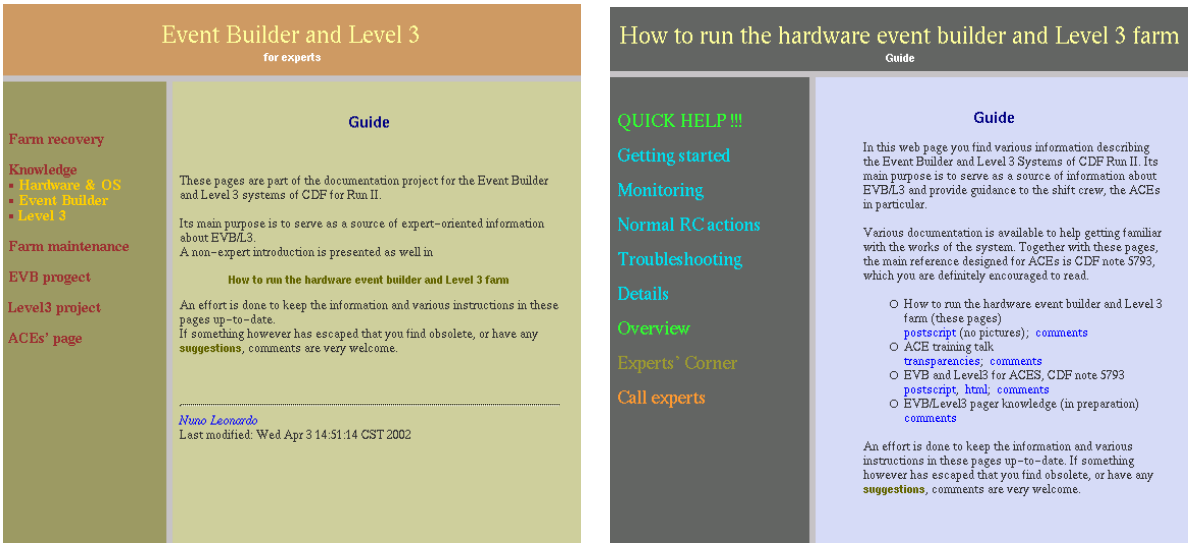


Figure 26: The EVB/L3 help pages for aces and pagers.

How to update these pages

Login to one of the online machines, and as cdface user go to the proper EVB/L3 directory,

```
  ...  > ssh b0dap30
b0dap18> su - cdface
b0dap18> cd /www/evbl3shift/
```

Changes may be committed to CVS. [87] These help pages reside in the *cdfevb_ace* package of CDFII online repository.

Web support-iii **CDF online home page**

The CDF Online Computing home page is located at

$$http://www-cdfonline.fnal.gov/$$

which contains various useful information.

Web support-iv **Linux documentation**

Linux is well documented on the web, both at FNAL [88] and elsewhere. [89]

[87] See CVS paragraph in a previous section.
[88] http://www-oss.fnal.gov/projects/fermilinux/
[89] E.g. http://www.tldp.org/

5.4 Trigger Manager

Trigger Manager-i **What it does, where it is running**

Trigger Manager (TM) is a process running on the Trigger Supervisor (TS), which is responsible for forwarding new L2 accepts to the Scanner Manager (SM) via SCRAMNet.

Recall, when 'done' signals are received by the TS from all FE crates, the TS communicates to TM, and this to the SM which coordinates (via SCRAMNet) the transfer of event data from SCPUs to a unique Converter.

It is a multi-task process, supporting eight partitions.

Trigger Manager-ii **TM tasks, scanFIFO**

The Trigger Supervisor crate includes a bus master CPU (*b0tsi00*, where TM runs), eight logic modules, each associated with a specific partition, as well as a SCRAMNet module. A given TS logic module (say, associated with partition N) possesses a scanFIFO, being capable of holding information associated with 16 events. TM reads Level2 accept from scanFIFO sending it to SM. Whenever scanFIFO is full no new trigger can be accepted.

We refer here the two tasks of importance in this context: the TM-SM poller, *tm_SM_pol* (associated with SCRAMNet communication), and the task talking to the scanFIFO associated with an existing partition, *tpn_#* (task partition number #, # being the identifier of the partition).

Logging to the TS one should make sure that at least these two tasks are present (except after RESET transition).

```
evbproxy@b0l3gate1> vxlogin b0tsi00
b0tsi00-> i

  NAME        ENTRY       TID    PRI   STATUS      PC       SP     ERRNO  DELAY
---------- ------------ -------- --- ---------- -------- -------- ------- -----
...
tm_SM_pol  tm_SM_pollst   b92c78 200 READY        2834ec   b92b68       0     0
tpn_0      tm_tsEach      b52a70 200 READY        201438   b52820  3d0002     0
...
value = 0 = 0x0
b0tsi00-> logout
```

If the tasks don't exist or have SUSPENDED status, the way to recover is by rebooting TS (need to make sure that no partitions are being used by nobody else).

Trigger Manager-iii **How to check communications between TM and SM (dumpEvb)**

Communication between the TM and the SM occur via SCRAMNet, for which are reserved the memory regions correspondent to identifiers **49** ($SM \rightarrow TM$) and **50** ($TM \rightarrow SM$). These can be checked using [90] the **dumpEvb(#)** tool (where # denotes the memory region identifier, i.e. 49 or 50 for SM-TM communications). [91]

[90] On any SCPU but not on SM.

[91] See SCRAMNet paragraph in a previous section.

5.5 DAQMON

DAQMON-i **How to start**

The DAQMON gui is started as follows,

```
b0dap#:~>  setup fer
b0dap#:~>  daqmon &
```

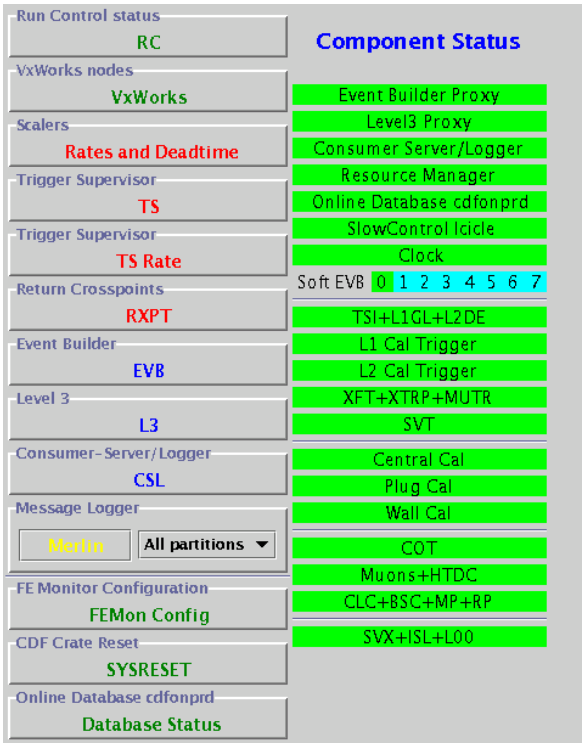


Figure 27: The DAQMON gui.

DAQMON-ii **Know what all tools are for**

Do it.

5.6 CSL

CSL-i **What is CSL, where it is physically** [92]

The *Consumer-Server/Logger* (CSL) is the system responsible for transferring the data received from Level3 to permanent storage. The events provided by Level3 (or by Software Event Builder) are made available to *consumers* [93] and sent over a high speed fiber link to the Feynman Computing Center (FCC).

CSL runs on b0dau32, [94] a dedicated server located on the third floor of B0 as well.

CSL-ii **How it is connected to L3**

The Level3 farm Output nodes, as well as Software Event Builder and consumer processes, are clients that connect to the CSL, which is a server. While the Output nodes and b0dau32 have an Ethernet connection, inside CSL processes communicate by means of message queues.

Physically, the output switch of the L3 farm Ethernet network, which receives connections from the Output nodes, contains 8 connections to CSL.

CSL-iii **What are the main components of CSL, explain message queues, receiver, logger, disk management, etc on a vague general level**

There are *receiver* processes responsible for receiving events from Level3; *consumer interface* processes, each associated with a consumer and responsible for sending events to that consumer; *logger* processes, one per partition, which write events to disk; *managing processes* (distributer) for disk management. These processes communicate via message queues.

CSL-iv **How to check CSL state using monitoring tools**

The CSL monitoring display (obtained from daqmon gui) provides various information about receiving, logging and sending events to consumers, the status of internal buffers (message queues button). It displays information for each receiver process: client node name, partition, number of events received, and rate. [95]

CSL-v **When to request restart of CSL**

When one notices that Level3 is presenting output related problems (e.g., many nodes appear green in the Level3 Display) it may be related to CSL not being accept events at the expected rate. One should check on the CSL monitoring displays whether events are being received by CSL, and being written to disk.

[92] Details can be found on the CSL help page, http://www-cdfonline.fnal.gov/ace2help/csl/

[93] Consumers are online monitoring processes, which look at a fraction of the events passing through CSL; these are followed online by the shift crew, namely by the *Consumer Operator, CO*.

[94] b0dau31 functions as a backup.

[95] Can be used e.g. to check if CSL is receiving events from a particular Output node, say.

If it is found necessary the CSL can be stopped and restarted. One (generally the ACE) needs to connect to b0dau32 as user ace (from b0dap53, 58, 59 as user cdfdaq), and from there execute *cslcom* followed by *check/stop/start/cleanup*. [96]

[96] See CSL help page for details and up-to-date procedure.

5.7 L3 Manager

L3 Manager-i **What does it do, where is it running**

Level3 manager is a SmartSockets client which during Cold/Warm Start provides the Level3 farm with the most recent calibrations (*calib* tarball).

It is maintained by the filter group, and runs on b0dap31. To check the necessary processes or execute the *l3manager_start* script one should log in to b0dap31 as *level3* user. [97]

Details about the sequence of relevant actions are as follows: i) l3manager receives Cold/Warm Start transition command from RC, ii) l3manager creates *calib* tarball (if needed), iii) l3manager replies to RC with information about which calib tag to use, iv) RC sends *L3ReadoutList* to L3, containing tags for *filter, calib, tcl.*

[97] Check the Level3 Manager Information page http://www-cdfonline.fnal.gov/ace2help/l3Manager/ for instructions.

5.8 SmartSockets

SmartSockets-i **What is SmartSockets, DaqMsg, Merlin**

SmartSockets [98] is the general CDF online/DAQ messaging system. It's a commercial product [99] (also UPS supported) used for processor (e.g., Linux, VxWorks) communications. It is based on a centralized server architecture, all messages being sent to a central process – the *RTServer* – which forwards them appropriately.

DaqMsg defines a standard format of the SmartSockets messages sent by CDF DAQ components. This is a product developed at CDF and found in ups and online cvs repository.

Merlin is a wrapper for DaqMsg. It defines standard DAQ messages which are circulated specifically between Run Control, Error Logger and DAQ systems, that are related to data taking.

SmartSockets-ii **What is RTServer**

RTServer is the central DAQ messaging process, and runs on b0dau30.

SmartSockets-iii **Where SmartSockets are used on Level3 farm**

The communications between the Level3 farm, i.e. b0l3pcom2 (or more precisely, the L3 proxy and *l3_mon_client* processes running there), and the RTServer are established via SmartSockets (SS). [100]

SmartSockets-iv **What happens if RTServer dies, if it is restarted**

If the RTServer hangs (e.g., number of connection licenses is exceeded) it needs to be restarted. One (generally the ACE) should log into b0dau30 as cdfdaq and kill the correspondent process (rtserver); normally it will be automatically restarted afterwards; otherwise it can be restarted from scratch (*setup smartsockets; rtserver*).

Once RTServer is restarted (or if b0dau30 has been rebooted, for that matter) one may need to restart the EVB and Level3 proxies.

[98] Details can be found on the help page http://www-cdfonline.fnal.gov/ace2help/acesmartsockets.html
[99] There is a limited number of connection licenses.
[100] See the paragraph on *L3 monitoring* above.

5.9 Oracle database

Oracle database-i **Where is it**

Our Oracle database is a software product. The primary database is always running on b0dau30. The actual programs running are called *instance of database*. This primary database we refer to as *Online Production instance of Oracle database*.

The database server b0dau30 has a special system of disk arrays attached to it, where the database information is stored. This disk storage is different from plain hard drives by its reliability and by the fact that everything is duplicated (mirrored). This is natural as we can not take any data when database is down. There is also a *integration instance* of the online database, which is completely separate, running on b0dau36, and is used for tests/development.

The primary, production instance of the DAQ online database has several logical sections serving different purposes. Most commonly known are: hardware database, calibration database and trigger database. Hardware database contains the inventory of all hardware of DAQ (cards, crates, PCs) and of all interconnections between the components. Calibration database has the results of all calibration runs for detector readout electronics. Trigger database defines the L1/L2/L3 trigger tables, specifying which triggers should be used in data taking with what parameters (e.g., single-lepton > 4 GeV, etc), what filter executables to be run on L3 farm, etc.

Oracle database-ii **How do we use it**

Both EVB and L3 proxies talk to the Oracle online production database when they are started, for obtaining the architecture information of EVB/L3.

Also, one sometimes needs to modify the hardware database entries by running the hardware database selector (*CardEditor*). [101]

Oracle database-iii **Is it okay if it is shut down, the effect on EVB/L3 operations**

If online production database is down, any attempt to restart EVB/L3 proxies will fail. It is sometimes possible to switch to online integration database by changing some environment variables for EVB and database connection information in l3proxy source code (and rebuilding L3 proxy).

Advanced

Oracle database-iv **Connecting to database and applying SQL**

It is possible to log into database, find necessary information and make appropriate changes. [102]

[101] See previous paragraph Hardware database for EVB.

[102] One needs to have a database account; and be careful with any changes to be performed.

Our Oracle database is based on SQL language.

An example follows (on any online PC). [103]

```
setup fer
sqlplus user@cdfonprod
@your_script.sql
commit;
exit;
```

[103] It is strongly recommended to verify any and all SQL scripts with knowledgeable experts.

APPENDIX

A APPENDIX

A.1 Help information on VxWorks commands

```
-> help

help                          Print this list
dbgHelp                       Print debugger help info
nfsHelp                       Print nfs help info
netHelp                       Print network help info
spyHelp                       Print task histogrammer help info
timexHelp                     Print execution timer help info
h         [n]                 Print (or set) shell history
i         [task]              Summary of tasks' TCBs
ti        task                Complete info on TCB for task
sp        adr,args...         Spawn a task, pri=100, opt=0, stk=20000
taskSpawn name,pri,opt,stk,adr,args... Spawn a task
td        task                Delete a task
ts        task                Suspend a task
tr        task                Resume a task
d         [adr[,nunits[,width]]] Display memory
m         adr[,width]         Modify memory
mRegs     [reg[,task]]        Modify a task's registers interactively
pc        [task]              Return task's program counter
version                       Print VxWorks version info, and boot line

Type <CR> to continue, Q<CR> to stop:

iam       "user"[,"passwd"]     Set user name and passwd
whoami                          Print user name
devs                            List devices
cd        "path"              Set current working path
pwd                             Print working path
ls        ["path"[,long]]     List contents of directory
ll        ["path"]            List contents of directory - long format
rename    "old","new"       Change name of file
copy      ["in"][,"out"]    Copy in file to out file (0 = std in/out)
ld        [syms[,noAbort][,"name"]] Load stdin, or file, into memory
                                (syms = add symbols to table:
                                -1 = none, 0 = globals, 1 = all)
lkup      ["substr"]          List symbols in system symbol table
lkAddr    address               List symbol table entries near address
checkStack  [task]              List task stack sizes and usage
printErrno  value               Print the name of a status value
period    secs,adr,args... Spawn task to call function periodically
repeat    n,adr,args...    Spawn task to call function n times (0=forever)
diskFormat  "device"          Format disk
diskInit  "device"            Initialize file system on disk
```

```
squeeze   "device"          Squeeze free space on RT-11 device

NOTE:  Arguments specifying 'task' can be either task ID or name.
```

A.2 Minicom keys

```
            Commands can be called by CTRL-A <key>

                Main Functions                  Other Functions

     Dialing directory..D  run script (Go)....G | Clear Screen.......C
     Send files.........S  Receive files......R | cOnfigure Minicom..O
     comm Parameters....P  Add linefeed.......A | Suspend minicom....J
     Capture on/off.....L  Hangup.............H | Exit and reset.....X
     send break.........F  initialize Modem...M | Quit with no reset.Q
     Terminal settings..T  run Kermit.........K | Cursor key mode....I
     lineWrap on/off....W  local Echo on/off..E | Help screen........Z
                                                | scroll Back........B

          Select function or press Enter for none.

                Written by Miquel van Smoorenburg 1991-1995
                Some additions by Jukka Lahtinen 1997-1998
                i18n by Arnaldo Carvalho de Melo 1998
```

A.3 DAQ VRB output data format

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%                                                    %%%%%%%%%%
%%%%%%%%%%                  RAW EVENT                       %%%%%%%%%%
%%%%%%%%%                                                    %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

This is an explanation of all words in raw event from
the wedge taken on Aug. 19, 1999. Consists of 1 SCPU,
1 VRB with only 1 link.

 6b6b1919       Sender ID                                          --------
      1f0       Message length                                      Message
 1f56be00       Message type                                        header
 33fafb00       Receiver ID                                        --------

 6c010000       Event length in bytes

 68010000       SCPU length in bytes                               --------
```

```
 1000000       ?                                                    SCPU
21000000       ? some SCPU information,                             header
43000012       ? not used by reformatter                            section
70380055       ? at the moment
76635f6d       ?                                                   --------

       0       VRB byte count = 0 __\ 64 bit                       --------
50010000       VRB byte count       /  word
       0       status of VRB                                        VRB
       0       \     Pointer section I*2 of VRB (link length)      header
       0        \  Total are 10 links. Here we see the last        section
       0         | 10th link is not 0, byteswap(3001)=>130=>4c,
       0        /  as seen in the next word.
30010000       /                                                   --------

4c000000       Number of words in current link                     --------
                                                                    |
444d4543       Bank name CEMD                                       |  ----
 1000000       Bank number (Oxuuuullll), where uuuu - total         |   |
                 number  of banks and llll - instance               |   |
       0       Bank next (for raw data always 0 as no two           |  Bank
                 versions of a bank                                 |   |
15000000       Bank length in words (32bits). Starts at 0 for       |  header
                 this word                                          |   |
 3000000       Bank type (1=I*2, 3=I*4, 4=R*4)                      |  ----
2c004005 2e004105 2b004205 2b004305  \                              |
30004405 32004505 2f004605 33004705   \                             |
2d004805 2e004905 2e004a05 2d004b05    |  Data                      |  Data
31004c05 2e004d05 2c004e05 2e004f05   /                             |
2d005005 2b005105 2f005205 30005305  /                              |
                                                                    |  ----
44414843       Bank name CHAD                                       |   |
 1000000                                                            |  Bank
       0       Bank header is the same as above                     |  header
29000000                                                            |   |
 3000000                                                            |  ----
27004005 36004105 2c004205 2a004305 2e004405 2d004505 \             |
31004605 2f004705 31004805 3f004905 2c004a05 1d004b05  \           Link
36004c05 34004d05 30004e05 2c004f05 32005005 29005105   |           |
30005205 26005305 2d008005 30008105 2f008205 30008305   | Data      |  Data
2c008405 34008505 2d008605 2f008705 2c008805 31008905   |           |
2c008a05 2f008b05 2b008c05 2d008d05 2c008e05 2e008f05  /            |
33009005 31009105 32009205 31009305                   /             |
                                                                    |  ---
       0       < Control word 0                                     |
aaaaaaaa       < Control word A                                     | Link
55555555       < Control word 5                                     | control
 2011700       < Tracer word (not checked by reformatter)            | words
```

```
80       < End of data word                                  |
 0       < possible pad word, for 64 bit alignment          -------
```

Acknowledgements

The author is supported by Fundação para a Ciência e a Tecnologia, Portugal, grant PRAXIS XXI/BD/18571/98, and a K.T. Kompton Fellowship, MIT.

References

[1] The CDF II Detector Technical Design Report, The CDF II Collaboration, FERMILAB-PUB-96/390-E

[2] EVB and Level3 for Aces, CDF/DOC/LEVEL-3/PUBLIC/5793; Ace help page for EVB/L3, How to run the hardware event builder and Level 3 farm, "http://www-cdfonline.fnal.gov/evbl3shift/evbl3shift.html"

[3] CDF Online Computing Home Page, "http://www-cdfonline.fnal.gov/"; DAQ Ace Information Web Page, " http://www-cdfonline.fnal.gov/ace2help/daqacehelp.html", and links therein.

[4] Additional EVB/Level3 documentation. Event builder messages, S.Tether; Letter of Intend for Run IIb: Upgrade of the Event-Builder and Level-3 Trigger PC Farm, CDF/DOC/LEVEL-3/CDFR/5516; Event-Building and PC Farm based Level-3 Trigger at the CDF Experiment, CDF/PUB/LEVEL-3/PUBLIC/5051; ATM Based Event Building and PC Based Level 3 Trigger at CDF, FERMILAB-CONF-98/348-E, CDF/PUB/ONLINE/PUBLIC/4789; Proposal for Expansion of the Event-Builder and Level-3 Trigger to a 1000 Hz Input Rate, CDF/DOC/LEVEL-3/PUBLIC/4623; Prototype of a PC Farm for the CDF Run II Level-3 Trigger, CDF/DOC/LEVEL-3/PUBLIC/4551

[5] Beginning Linux programming, R.Stones, N.Matthew, Wrox Press, Chicago; The Linux Documentation Project, "http://www.tldp.org/"; UPS/UPD v4 Product Page at FNAL, "http://www.fnal.gov/docs/products/ups/"; CVS page, "http://www.cvshome.org/", "http://www.cvshome.org/docs/manual/cvs.html"

[6] VxWorks Reference Manual, 5.4, Wind River Systems, Inc., 1999, "http://www.windriver.com/pdf/ref.pdf"

[7] SCRAMNet+ Network, VME6U Hardware Reference, Systran Inc., 1998; SCRAMNet Network, PCI Bus Hardware Reference, Systran Inc., 1998; "http://www.systran.com/scmain.html"